UNIVERSIDADE DO VALE DO ITAJAÍ
CENTRO DE CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ANÁLISE DE TÉCNICAS E METODOLOGIAS DE DESENVOLVIMENTO DE SOFTWARE PARA SISTEMAS COMPUTACIONAIS EMBARCADOS

Área de Sistemas da Computação

por

Cristiane Gomes Bim dos Santos

Cesar Albenes Zeferino, Dr.
Orientador

Itajaí (SC), novembro de 2005

UNIVERSIDADE DO VALE DO ITAJAÍ
CENTRO DE CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ANÁLISE DE TÉCNICAS E METODOLOGIAS DE DESENVOLVIMENTO DE SOFTWARE PARA SISTEMAS COMPUTACIONAIS EMBARCADOS

Área de Sistemas Embarcados

por

Cristiane Gomes Bim dos Santos

Relatório apresentado à Banca Examinadora do Trabalho de Conclusão do Curso de Ciência da Computação para análise e aprovação.
Orientador: Cesar Albenes Zeferino, Dr.

Itajaí (SC), novembro de 2005

SUMÁRIO

LISTA DE ABREVIATURAS ..... **iv**
LISTA DE FIGURAS ..... **v**
LISTA DE TABELAS ..... **vii**
RESUMO ..... **viii**
ABSTRACT ..... **ix**
1. INTRODUÇÃO **1**
1.1. OBJETIVOS **2**
1.1.1. Objetivo Geral **2**
1.1.2. Objetivos Específicos **2**
1.2. METODOLOGIA **2**
1.3. ESTRUTURA DO TRABALHO **3**
2. FUNDAMENTAÇÃO TEÓRICA **4**
2.1. SISTEMAS EMBARCADOS **4**
2.1.1. Definição **4**
2.1.2. Aplicações dos Sistemas Embarcados **4**
2.1.3. Projeto de Sistemas Embarcados **5**
2.1.4. Sistemas Embarcados de Tempo Real **6**
2.1.5. Microprocessadores e Microcontroladores no Projeto de Sistemas Embarcados **8**
2.2. TÉCNICAS E METODOLOGIAS DE DESENVOLVIMENTO **9**
2.3. METODOLOGIA ROPES **10**
2.3.1. Análise **11**
2.3.2. Projeto **17**
2.3.3. Implementação **21**
2.3.4. Teste **21**
2.4. METODOLOGIA WOLF **22**
2.4.1. Requisitos **23**
2.4.2. Especificação **24**
2.4.3. Arquitetura **27**
2.4.4. Projeto dos Componentes **28**
2.4.5. Integração do Sistema **28**
2.5. MEDIDAS DE SOFTWARE **29**
2.5.1. Medida orientada ao tamanho **30**
2.5.2. Medida orientada à função **31**
2.6. O MODELO CMMI **32**
3. DESENVOLVIMENTO **34**

3.1. APLICAÇÃO 1: TESTADOR DE DISPLAY DE 7 SEGMENTOS 35
3.1.1. Requisitos da Aplicação 1 35
3.1.2. Projeto da Aplicação 1 41
3.1.3. Implementação e Teste da Aplicação 1 42
3.2. APLICAÇÃO 2: CRONÔMETRO 43
3.2.1. Requisitos da Aplicação 2 44
3.2.2. Projeto da Aplicação 2 52
3.2.3. Implementação e Teste da Aplicação 2 54
3.3. APLICAÇÃO 3: MONITORAMENTO DE DADOS REMOTOS 55
3.3.1. Requisitos da Aplicação 3 56
3.3.2. Projeto da Aplicação 3 70
3.3.3. Implementação e Teste da Aplicação 3 72
3.4. DEFINIÇÃO DAS MEDIDAS PARA ANÁLISE 74
3.5. ANÁLISE DAS METODOLOGIAS 75
4. CONCLUSÕES 78
REFERÊNCIAS BIBLIOGRÁFICAS 80

LISTA DE ABREVIATURAS

CI	Circuito Integrado
CMMI	Capability Maturity Model Integration
CPU	Central Processing Unit
DFD	Diagrama de Fluxo de Dados
E/S	Entrada/Saída
LCD	Liquid Cristal Display
LED	Light Emitter Diode
LOC	Lines of Code
LSED	Laboratório de Sistemas Embarcados e Distribuídos
MEF	Máquina de Estado Finito
PC	Personal Computer
RAM	Random Access Memory
ROPES	Rapid Object-Oriented Process for Embedded System
SEI	Software Engineering Institute
TCC	Trabalho de Conclusão de Curso
UML	Unified Modeling Language
UNIVALI	Universidade do Vale do Itajaí

LISTA DE FIGURAS

Figura 1. Arquitetura típica de um sistema embarcado....6
Figura 2. Ciclo de vida do software....9
Figura 3. Notação usada pelo diagrama de caso de uso....12
Figura 4. Exemplo de um caso de uso....13
Figura 5. Exemplo de um diagrama de classe....15
Figura 6. Exemplo de um diagrama de seqüência....17
Figura 7. Exemplo de um diagrama de componentes....19
Figura 8. Passos para projeto de sistemas embarcados....23
Figura 9. Exemplo de uma máquina de estados finitos para um portão....26
Figura 10. Exemplo de um diagrama de estados....26
Figura 11. Exemplo de um fluxograma....27
Figura 12. Aplicação Testador de Display de 7 Segmentos....35
Figura 13. Diagrama de caso de uso da aplicação Testador de Display de 7 Segmentos....35
Figura 14. Diagrama de classes da aplicação Testador de Display de 7 Segmentos....36
Figura 15. Diagrama de seqüência da aplicação Testador de Display de 7 Segmentos por varredura....36
Figura 16. Diagrama de seqüência da aplicação Testador de Display de 7 Segmentos por interrupção....37
Figura 17. Diagrama de classes da aplicação Testador de Display de 7 Segmentos segundo WOLF....39
Figura 18. Diagrama de seqüência da aplicação Testador de Display de 7 Segmentos por varredura....39
Figura 19. Diagrama de estado da aplicação Testador de Display de 7 Segmentos....40
Figura 20. Fluxograma da aplicação Testador de Display de 7 Segmentos....40
Figura 21. Diagrama de componentes e construção da aplicação Testador de Display de 7 Segmentos....41
Figura 22. Diagrama de blocos do software da aplicação Testador de Display de 7 de Segmentos .41
Figura 23. Diagrama de blocos do hardware da aplicação Testador de Display de 7 de Segmentos 42
Figura 24. Esquema da ligação física do hardware da aplicação Testador de Display de 7 Segmentos....42
Figura 25. Trecho do código da Aplicação Testador de Display de 7 Segmentos....43
Figura 26. Aplicação Cronômetro....44
Figura 27. Diagrama de caso de uso geral da aplicação Cronômetro....44
Figura 28. Diagrama de caso de uso detalhado da aplicação Cronômetro....45
Figura 29. Diagrama de classes da aplicação Cronômetro....46
Figura 30. Diagrama de seqüência da aplicação Cronômetro....47
Figura 31. Diagrama de classes da aplicação Cronômetro segundo WOLF....49
Figura 32. Diagrama de seqüência da aplicação Cronômetro....50
Figura 33. Diagrama de estado da aplicação Cronômetro....51
Figura 34. Fluxograma da aplicação Cronômetro....52
Figura 35. Diagrama de componentes e construção da aplicação Cronômetro....53
Figura 36. Diagrama de blocos do software da aplicação Cronômetro....53
Figura 37. Diagrama de blocos do hardware da aplicação Cronômetro....53
Figura 38. Esquema de ligação física do hardware da aplicação Cronômetro....54
Figura 39. Trecho de código da aplicação Cronômetro I....55

Figura 40. Trecho de código da aplicação Cronômetro II.....55
Figura 41. Aplicação Monitoramento de Dados Remotos 56
Figura 42. Diagrama de caso de uso geral da aplicação Monitoramento de Dados Remotos 57
Figura 43. Diagrama de caso de uso detalhado da aplicação Monitoramento de Dados Remotos....58
Figura 44. Diagrama de caso de uso da aplicação Monitoramento de Dados Remotos 59
Figura 45. Diagrama de classes da aplicação Monitoramento de Dados Remotos.....60
Figura 46. Primeira parte do diagrama de seqüência da aplicação Monitoramento de Dados Remotos 61
Figura 47. Segunda parte do diagrama de seqüência da aplicação Monitoramento de Dados Remotos 62
Figura 48. Diagrama de classes da aplicação Monitoramento de Dados Remotos.....64
Figura 49. Primeira parte do diagrama de seqüência da aplicação Monitoramento de Dados Remotos 65
Figura 50. Segunda parte do diagrama de seqüência da aplicação Monitoramento de Dados Remotos 66
Figura 51. Diagrama de classes da aplicação Monitoramento de Dados Remotos.....67
Figura 52. Primeira parte do fluxograma da aplicação Monitoramento de Dados Remotos 68
Figura 53. Segunda parte do fluxograma da aplicação Monitoramento de Dados Remotos 69
Figura 54. Diagrama de componentes e construção da aplicação Monitoramento de Dados Remotos 70
Figura 55. Diagrama de blocos do software da aplicação Monitoramento de Dados Remotos.....71
Figura 56. Diagrama de blocos do hardware da aplicação Monitoramento de Dados Remotos.....71
Figura 57. Esquema da ligação física do hardware da aplicação Monitoramento de Dados Remotos 72
Figura 58. Código da função que efetua a leitura dos sensores 73
Figura 59. Trecho de código da função transmite dados.....73
Figura 60.Trecho de código da função de recepção de dados.....73
Figura 61. Trecho de código da aplicação Monitoramento de Dados Remotos 74

LISTA DE TABELAS

Tabela 1. Fases de projeto 19
Tabela 2. Medidas orientadas ao tamanho. 31
Tabela 3. Fases de Projeto 34
Tabela 4. Nível de viabilidade 76
Tabela 5. Nível de complexidade 76
Tabela 6. Nível de agilidade 77

RESUMO

BIM, Cristiane. **Análise de Técnicas e Metodologias de desenvolvimento de software para sistemas computacionais embarcados.** Itajaí, 2005. 62 f. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação)–Centro de Ciências Tecnológicas da Terra e do Mar, Universidade do Vale do Itajaí, Itajaí, 2005.

No mundo moderno, os sistemas computacionais estão presentes em um número cada vez maior de equipamentos, realizando tarefas de interfaceamento, controle e processamento. Tais sistemas, denominados sistemas embarcados, podem ser encontrados em eletrodomésticos, telefones celulares e automóveis. Considerando os fatos citados e sabendo que no Laboratório de Sistemas Embarcados Distribuídos (LSED) da Universidade do Vale do Itajaí, são desenvolvidos projetos de sistemas embarcados para os quais ainda não se têm identificado, claramente, a melhor metodologia de projeto a ser adotada, o presente trabalho pretende levantar subsídios através do estudo de técnicas e metodologias de projeto de sistemas, a fim de atender essa demanda com informações que auxiliem analistas e projetistas na escolha e na utilização da melhor técnica e/ou metodologia para o projeto de aplicações embarcadas baseadas em microcontroladores. Neste texto, é apresentada uma revisão bibliográfica sobre conceitos a respeito de sistemas embarcados, técnicas e metodologias para o desenvolvimento de tais sistemas e medidas de software. Em seguida, apresenta-se a modelagem de três aplicações utilizando-se as técnicas e metodologias de projeto estudadas. Essas aplicações são a base de referência para a análise comparativa das metodologias, onde buscou-se determinar subsídios que orientem a escolha dos procedimentos metodológicos para o projeto das classes de aplicação desenvolvidas no LSED.

Palavras-chave: Sistemas Embarcados. Microcontrolador. Metodologias de projeto.

ABSTRACT

In the modern world, the computing systems are present in a growing number of equipments, doing tasks of interfacing, control and processing. These systems, named embedded systems, can be founded into appliances, cell phones and automobiles. Considering the citied facts, and know that in the Embedded and Distributed Systems Laboratory (LSED), they are developed some embedded systems designs, for which it was not still identified the best design methodology, this work intent perform a study about the techniques and methodologies used in the design of computing systems, aiming to aid the analysts and designers of microcontroller-based embedded computing systems with the task of choose the best technique and/or methodology to be used. In this text, it is presented a review about concepts on embedded systems, design techniques and methodologies for the development of computing systems, and software metrics. After that, it is presented the modeling of three microcontroller-based applications, what was done by using the studied techniques and methodologies. Such applications will be implemented in the second phase of this research project, and will serve as a reference for the comparison of the used methodologies. With such analysis, it is intended to obtain the guidelines to help the chosen of the best design approach for the class of application typically developed at LSED.

***Keywords**: Embedded Systems. Microcontroller. Design Methodology.*

1. INTRODUÇÃO

Com o avanço tecnológico, os sistemas computacionais embarcados estão cada vez mais presentes na rotina das pessoas, seja na utilização de um forno microondas, de um carro, de um telefone celular, de uma impressora, entre muitos outros aparelhos.

O projeto desse tipo de sistema computacional é extremamente complexo, por envolver conceitos até agora pouco analisados pela computação de propósito geral. Por exemplo, as questões da portabilidade e do limite de consumo de potência sem perda de desempenho, a baixa disponibilidade de memória, a necessidade de segurança e confiabilidade, a possibilidade de funcionamento em uma rede maior, e o curto tempo de projeto tornam o desenvolvimento de sistemas computacionais embarcados uma área em si (WOLF, 2001).

Outro aspecto importante diz respeito ao tempo disponível para a execução do projeto de um sistema embarcado. Segundo Carro e Wagner (2003), a grande pressão mercadológica num mercado mundial globalizado, somada à contínua evolução tecnológica, impõe às empresas a necessidade de projetarem novos sistemas embarcados dentro de janelas de tempo cada vez mais estreitas, de poucos meses.

Apesar de possuir uma abrangência significativa, a área de Sistemas Embarcados ainda não possui um suporte metodológico consolidado que facilite a fase de análise e projeto de tais sistemas. Como exemplo, no contexto da UNIVALI, os pesquisadores do LSED (Laboratório de Sistemas Embarcados e Distribuídos) desenvolvem projetos na área de Sistemas Embarcados, os quais demandam técnicas e metodologias mais adequadas à classe de aplicações tipicamente desenvolvida (supervisão e controle).

A execução deste projeto de pesquisa se justifica como TCC para o Curso de Ciência da Computação, pois utiliza teorias, conceitos e tecnologias de disciplinas ministradas no curso, como Arquitetura de Computadores, Engenharia de Software e Programação, além de envolver tecnologias e conceitos complementares, como microcontroladores e sistemas embarcados.

1.1. OBJETIVOS

1.1.1. Objetivo Geral

O objetivo geral deste Trabalho de Conclusão de Curso é analisar técnicas e metodologias de desenvolvimento de software para sistemas computacionais embarcados, disponibilizando subsídios que auxiliem analistas e projetistas na escolha e na utilização de uma metodologia ou fases da mesma no desenvolvimento de software para sistemas computacionais embarcados.

1.1.2. Objetivos Específicos

Os objetivos específicos deste trabalho são:

1. Pesquisar e analisar metodologias já disponíveis para o projeto de sistemas embarcados;
2. Pesquisar e analisar metodologias e técnicas de desenvolvimento de software que possam ser aplicadas ao desenvolvimento de software para sistemas embarcados;
3. Definir medidas para avaliação das metodologias e técnicas analisadas;
4. Avaliar as metodologias e técnicas por meio do desenvolvimento de três aplicações, comparando o uso dessas metodologias e técnicas no desenvolvimento do software dessas aplicações;
5. Documentar e divulgar os resultados do trabalho desenvolvido.

1.2. METODOLOGIA

Primeiramente, fez-se um estudo sobre técnicas e metodologias para o projeto de sistemas computacionais, especialmente de sistemas embarcados, e de medidas de software. O estudo foi feito através de pesquisa bibliográfica de artigos disponibilizados na Internet e de livros especializados. Destacam-se como referências principais desse estudo os livros “Computer as Component” de Wayne WOLF (2001) e “Real-Time UML second edition” de Bruce Powel Douglass (2000).

Com as informações adquiridas no estudo foi elaborada a Fundamentação Teórica que apresenta as informações sobre as principais técnicas e metodologias existentes, bem como as medidas de software mais utilizadas. Uma apresentação sobre sistemas embarcados também foi elaborada.

O passo seguinte foi selecionar as três aplicações utilizadas como estudo de caso para a avaliação das técnicas e metodologias. Após a seleção das aplicações, foi então elaborado o modelo conceitual das mesmas, especificando o funcionamento de cada uma.

Com o estudo das medidas de software juntamente com o conceito do CMMI, foram definidas as medidas adotadas neste trabalho para a avaliação das metodologias aplicadas no desenvolvimento do software das aplicações.

Realizou-se a análise de cada aplicação utilizando os conceitos descritos na fundamentação teórica. Cada aplicação foi implementada e validada. Com os resultados dessa implementação, foi efetuada uma análise comparativa do processo de desenvolvimento do software dessas aplicações utilizando as medidas de avaliação definidas neste trabalho.

1.3. ESTRUTURA DO TRABALHO

O presente trabalho está dividido em quatro capítulos, sendo eles: Introdução, Fundamentação Teórica, Projeto e Conclusão. O Capítulo 2, Fundamentação Teórica, apresenta uma introdução a sistemas computacionais embarcados e uma síntese dos conceitos de técnicas e metodologias de projeto e medidas de software. No Capítulo 3, Projeto, é apresentada a descrição e o processo de análise, implementação e validação das aplicações selecionadas, finalizando com a avaliação das metodologias e técnicas utilizadas. O Capítulo 4 apresenta as Conclusões extraídas do trabalho e apresenta algumas oportunidades de pesquisa que foram observadas durante a elaboração do mesmo.

2. FUNDAMENTAÇÃO TEÓRICA

2.1. SISTEMAS EMBARCADOS

2.1.1. Definição

Zelenovsky e Mendonça (1999) definem um sistema embarcado como sendo um sistema computacional que foi incluído em um outro sistema com a finalidade outra que a de fornecer processamento genérico.

Um sistema embarcado contém um computador como parte de um sistema maior, fornecendo algum serviço específico ao usuário como por exemplo, aquecer uma refeição (DOUGLASS, 2000).

Ainda segundo Wayne Wolf (2001), um sistema embarcado é "qualquer dispositivo que inclui um computador programável, mas que não seja ele próprio um computador de propósito geral". Em suma, um sistema embarcado é qualquer computador que é componente de um sistema maior e que possui o seu próprio microprocessador.

Os sistemas embarcados também são conhecidos como sistemas embutidos, sistemas dedicados, sistemas integrados ou sistemas acoplados, sendo que na literatura em inglês são denominados por Wayne Wolf (2002) e outros autores como *embedded systems* ou *embedded computers*.

2.1.2. Aplicações dos Sistemas Embarcados

De acordo com Zelenovsky e Mendonça (1999), a evolução da microeletrônica e o barateamento das CPUs viabilizaram o emprego de sistemas computadorizados nos diversos equipamentos. Os cérebros da maioria dos equipamentos modernos são os pequenos computadores que eles trazem embutidos.

Segundo Carro e Vagner (2003),

> Os sistemas computacionais embarcados estão presentes em praticamente todas as atividades humanas e, com os baixos custos tecnológicos atuais, tendem a aumentar sua presença no cotidiano das pessoas. Exemplos de tais sistemas são os telefones celulares com máquina fotográfica e agenda, os sistemas de controle dos carros e dos ônibus, os

> computadores portáteis do tipo *palm-top*, os fornos de microondas com controle de temperatura inteligente, as máquinas de lavar e outros eletrodomésticos.

As aplicações de sistemas embarcados vão desde sistemas pequenos que requerem pouco processamento até sistemas complexos que exigem processamento crítico e alto desempenho com execução em tempo-real. Em todas elas, os sistemas computacionais embarcados caracterizam-se por serem dedicados a uma aplicação ou a uma pequena faixa de aplicações (MORAES *et al*., 2001).

Os sistemas embarcados estão em todo lugar, desde a máquina de lavar roupa e o microondas até o telefone, o som, a televisão, e o automóvel. Eles ajudam as pessoas a torrar o pão, a identificar pessoas ao telefone, entre outras facilidades utilizadas em seu cotidiano (DOUGLASS, 2000).

A CNZ Engenharia (2005) afirma que "a inteligência incorporada às máquinas está presente em todos os lugares, e a qualquer momento. Estima-se que, em 2010, em média uma pessoa interagirá com 350 dispositivos com microcontroladores diariamente".

2.1.3. Projeto de Sistemas Embarcados

O projeto de sistemas computacionais embarcados é mais abrangente que o projeto de sistemas computacionais de propósito geral pois, além de envolver todo o conceito de engenharia de software, envolve conceitos de microeletrônica, como gasto de potência, desempenho, portabilidade entre outros. Carro e Wagner (2003) afirmam que "o projeto de sistemas eletrônicos embarcados enfrenta diversos grandes desafios, pois o espaço de projeto arquitetural a ser explorado é muito vasto".

O mercado atual força a redução de custos e tempo de projeto, o que leva os projetistas a desejar deslocar a maior parte da funcionalidade dos sistemas embarcados para o software, deixando os elementos de hardware dedicados apenas para funcionalidades que necessitam de alto desempenho. Sendo assim, faz-se necessário direcionar esforços para analisar e conceber de forma automática a melhor distribuição das funções do sistema entre o software e o hardware (MORAES *et al*., 2001).

Segundo Oyamada e Wagner (1999),

> o projeto de sistemas embarcados complexos compreende um conjunto de diferentes tecnologias, ferramentas e estilos de projeto. Um ambiente de projeto para tais sistemas deve considerar a especificação do sistema, particionamento em software, hardware digital e partes analógicas, e síntese das partes de hardware e software.

A Figura 1 ilustra um típico sistema embarcado, apresentando os componentes básicos que o forma. A CPU é responsável pelo controle do sistema, ela faz a comunicação com a RAM, onde ficam os dados utilizados pelo sistema em seu funcionamento, com a ROM/Flash, onde ficam armazenadas as informações recolhidas pelo sistema, o Temporizador, que determina a velocidade de processamento do sistema, e finalizando, os dispositivos de entrada e saída, que fazem a comunicação o mundo externo.

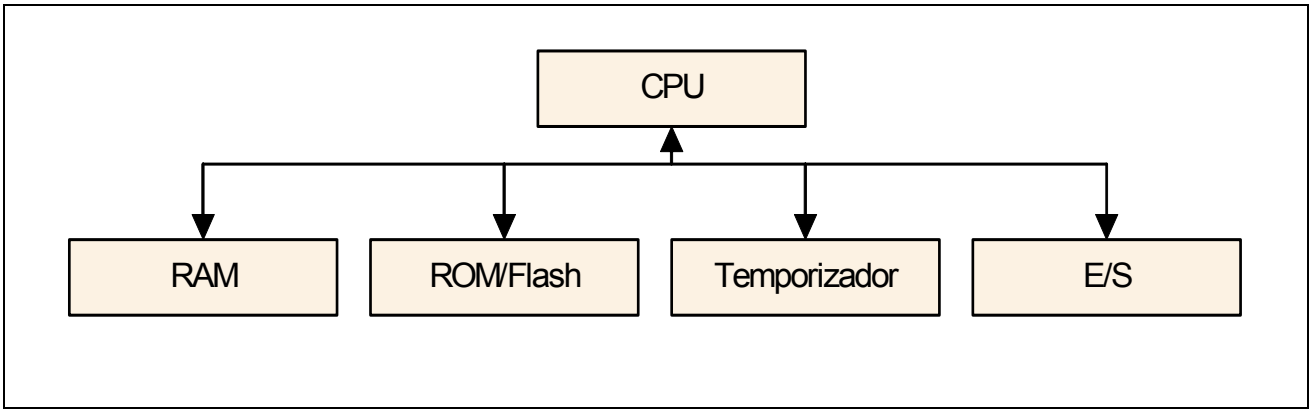

Figura 1. Arquitetura típica de um sistema embarcado
Fonte: Adaptado de Zelenovsky e Mendonça (1999).

2.1.4. Sistemas Embarcados de Tempo Real

Sistemas de Tempo-real, também chamados de sistemas reativos, são sistemas computacionais com a finalidade de monitorar, responder ou controlar um ambiente externo cuja conexão é feita através de sensores, atuadores e outras interfaces de entrada/saída. Sua função principal é responder ou reagir a sinais provenientes de seu ambiente. Como exemplo pode-se citar um sistema controlador de velocidade em um automóvel (SHAW, 2003).

O software desses sistemas embarcados é mais difícil de construir que os softwares para microcomputadores do tipo *desktop*. Os sistemas de tempo-real possuem todos os problemas das aplicações para *desktop* adicionados de muitos outros. Muitas vezes, não possuem o monitor de um

computador convencional ou o teclado, mas estão no controle de alguns dispositivos que aparentemente não são computadorizados (DOUGLASS, 2000).

Segundo Galvin Silberchatz (2000), um sistema de tempo-real é usado quando existem requisitos rígidos relativos ao tempo sobre a operação de um processador ou sobre o fluxo de dados e é assim freqüentemente usado como um dispositivo de controle em uma aplicação dedicada a um propósito específico.

Alan C. Shaw (2003) explica que software de tempo-real difere significativamente de software convencional de diversas maneiras e apresenta as principais diferenças descritas a seguir.

A primeira diferença é a importância das restrições temporais, pois além de produzir a resposta ou saída correta, o programa deve fazer isso dentro de um prazo estipulado. Na maioria dos casos, o software de tempo-real deve também satisfazer condições de tempo, as quais envolvem relações sobre tempos relativos e absolutos. A mais comum e simples de tais condições é o prazo-limite (*deadline*) – um limite sobre o tempo absoluto ou relativo no qual o resultado deve ser obtido.

A segunda diferença diz respeito à concorrência. Os sistemas de tempo-real, além das concorrências comuns utilizadas pelos sistemas de computação, para melhorar o desempenho ou para representar atividades paralelas, devem também atender as concorrências físicas relacionadas ao mundo externo com o qual estão conectados, pois sinais vindos do ambiente externo podem chegar simultaneamente.

Uma terceira característica importante é a importância de confiabilidade e tolerância a falhas. Confiabilidade é a medida de quão freqüentemente um sistema falhará, pois virtualmente nenhum sistema é perfeitamente confiável e falhas devem ser esperadas. Erros e falhas podem ser muito caros, causando até a perda de vida humana. Conseqüentemente, é muito importante evitar falhas, se possível, através de técnicas para incrementar a confiabilidade e responder de forma apropriada com o menor custo possível às falhas que efetivamente venham a ocorrer.

Uma diferença final apresentada por Alan C. Shaw (2003) entre programas de tempo-real e programas convencionais relaciona-se ao teste e a certificação. Por causa dos altos custos associados com as falhas, usualmente não é possível testar e depurar sistemas em seus ambientes reais completos. Ao invés disso, deve-se confiar em simulações, testes de subsistemas,

especificações cuidadosas, análise de projetos abrangentes e procedimentos de execução para detecção e tratamento de falhas.

2.1.5. Microprocessadores e Microcontroladores no Projeto de Sistemas Embarcados

A CNZ Engenharia (2005) explica que a evolução rápida da eletrônica digital, dos microprocessadores e, em particular, dos microcontroladores provocou uma revolução no cotidiano das pessoas. Nos afazeres domésticos diários, na condução de um veículo, no cenário visual da cidade e, também nos mais variados equipamentos que estão a nossa disposição no trabalho ou na escola encontram-se soluções integradas (ou *embedded*) que utilizam microcontroladores.

Segundo Wayne Wolf (2001), o uso de microprocessadores no projeto de sistemas embarcados se justifica por duas razões: a primeira delas é que eles constituem o caminho mais eficiente para implementar sistemas digitais, e a segunda é que os microprocessadores facilitam o projeto de famílias de produtos.

Descrevendo o espaço do microprocessador em sistemas embarcados, Carro e Wagner (2003) afirmam que

> o projeto de sistemas embarcados toma sempre como base um ou mais processadores. Embora essa solução pareça extremamente conservadora do ponto de vista de inovação, ela traz enormes vantagens do ponto de vista operacional. Primeiro, o fator de escala. Como os microprocessadores são encontrados em milhares de projetos, seu custo dilui-se entre muitos clientes, as vezes até competidores entre si. Mais ainda, uma vez que uma plataforma baseada em processador esteja disponível dentro de uma empresa, novas versões de produtos podem ser feitas pela alteração do software da plataforma. A personalização do sistema dá-se através do software de aplicação, que toma atualmente a maior parte do tempo de projeto. Além dessas vantagens competitivas, há ainda o fator treinamento de engenheiros, já que estes geralmente se formam com conhecimentos de programação de microprocessadores.

O desenvolvimento dos microcontroladores deu-se ao ser possível colocar o processador e os periféricos no mesmo circuito integrado. Esses dispositivos são componentes baseados na arquitetura dos microprocessadores e podem ser usados em diversos projetos de eletrônica, substituindo muitos componentes digitais e melhorando o acabamento e o desempenho do projeto, principalmente devido à redução do espaço físico ocupado e ao aumento da eficiência (SOUZA, 2000).

Segundo Carlos Cimolin (2004), os microcontroladores possuem todos os componentes necessários para o controle de um processo, desde a memória interna onde fica o programa, uma memória de dados, portas de entrada e ou saída e diversos periféricos. Essa é uma das principais características que diferenciam os microcontroladores dos microprocessadores, nos quais os periféricos não são incluídos na mesma pastilha da CPU (Central Processing Unit ou Unidade Central de Processamento).

Ainda, segundo Zelenovsky e Mendonça (1999), com o avanço da microeletrônica, o microcontrolador recebeu uma quantidade de recursos cada vez maior dentro do seu CI (Circuito Integrado), levando a primeira definição de microcontrolador, como sendo um CI com alta densidade de integração que inclui, dentro do chip, a maioria dos componentes necessários para o controlador.

2.2. TÉCNICAS E METODOLOGIAS DE DESENVOLVIMENTO

Segundo Alan C. Shaw (2003), a área de Engenharia de Software preocupa-se com as técnicas, metodologias e processos para a construção de software e explica que o "ciclo de vida" do software define os estágios de um sistema de software à medida que ele se desenvolve, a partir de sua especificação inicial até sua instalação e utilização. O ciclo de vida do software diagramado na Figura 2 normalmente é dividido em cinco fases.

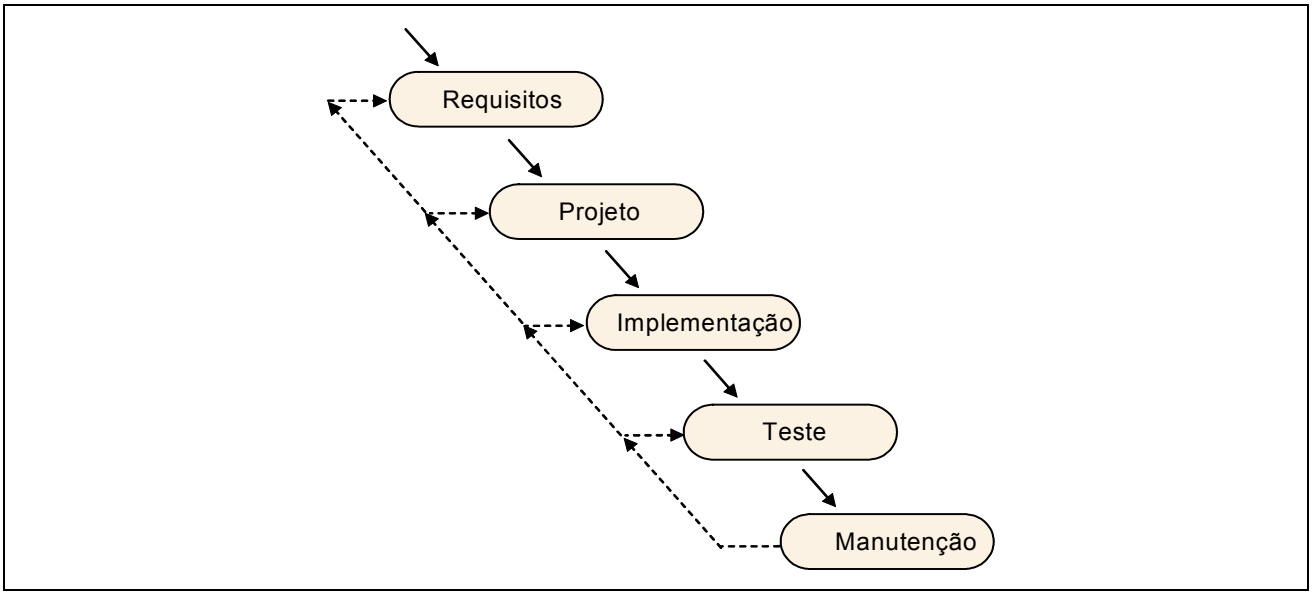

Figura 2. Ciclo de vida do software
Fonte: Shaw (2003)

A fase de requisitos determinará o que o software deverá fazer, sendo elaborada em conjunto com o cliente. O contratante é responsável por fornecer o software para o cliente. Requisitos descrevem um acordo ou contrato entre as duas partes, especificando o que o software deve fazer para ser aprovado em um teste de aceitação.

A fase de projeto preocupa-se tanto com o projeto detalhado ou de baixo nível, quanto com o projeto arquitetônico. Questões de baixo nível compreendem as definições de algoritmos e de estruturas de dados, além do projeto detalhado de partes individuais, como sub-rotinas, módulos e arquivos. O nível arquitetônico mais alto enfatiza a definição, a organização e o relacionamento dos objetos ou módulos de software que poderiam ser empregados para atingir os requisitos. Essa fase também enfoca as interfaces dos componentes arquitetônicos.

Ao ser atingido o estágio de implementação, decisões de linguagem de programação e de sistema operacional já foram tomadas. O software está agora realmente codificado. A principal tarefa intelectual é construir os módulos de programa e suas interfaces detalhadas.

O componente de teste do processo de Engenharia de Software corresponde ao teste de integração. Componentes são combinados incrementalmente para verificar se cada subsistema resultante satisfaz as propriedades globais requeridas durante a execução.

Muitas fases do ciclo de vida do software, em parte, são repetidas durante a manutenção, após ele estar em uso. Melhoramentos na tecnologia de hardware, novos desenvolvimentos teóricos, a disponibilidade de melhor software de suporte, a descoberta de erros e as mudanças em necessidades de clientes podem resultar em uma demanda contínua de mudanças no sistema.

Nas seções a seguir, são apresentadas as metodologias para o processo de desenvolvimento de sistemas embarcados disponíveis na literatura e que foram selecionadas para a realização da análise das mesmas.

2.3. METODOLOGIA ROPES

A metodologia ROPES (Rapid Object-Oriented Process for Embedded Systems), apresentada por Bruce Powel Douglass (2000), é um processo efetivo de aplicação da UML (Unified Modeling Language) no desenvolvimento de aplicações de tempo-real e sistemas embarcados.

A UML é uma linguagem padrão para a elaboração da estrutura de projetos de software, sendo normalmente empregada para a visualização, a especificação, a construção e a documentação de artefatos que façam uso de sistemas complexos de software. Por ser apenas uma linguagem, a UML é somente uma parte de um método para desenvolvimento de software (JACOBSON, 2000).

A UML é um modelo de linguagem, não um método. Um método pressupõe um modelo de linguagem e um processo. O modelo de linguagem é a notação que o método usa para descrever o projeto. O processo são os passos que devem ser seguidos para se construir o projeto (ESMIN, 2005).

Segundo Grant Martin (2002), UML não é uma simples linguagem, mas um conjunto de notações, sintaxes e semânticas permitindo a criação de famílias de linguagens para aplicações particulares, sendo uma "meta-linguagem".

As principais atividades utilizadas pelas metodologias de projeto de software em uso hoje em dia também são utilizadas pelo processo ROPES, sendo elas: Análise, Projeto, Implementação e Teste.

2.3.1. Análise

Nesta fase são capturadas as necessidades do usuário e o comportamento do sistema através da análise de casos de uso, também conhecido por "*Use Case*". As entidades externas ao sistema, em UML chamados de "atores externos", que interagem com o sistema são modelados entre as funções que eles requerem.

A análise define as propriedades essenciais da aplicação, levantando os conceitos e estruturas chaves no sistema independentemente de como a solução será implementada. As principais fases da análise são: a análise de requisitos e a análise de objetos.

2.3.1.1. Análise de Requisitos

Uma das primeiras coisas a ser feita em um projeto é a determinação dos requisitos detalhados do sistema. Esses requisitos podem ser as capacidades do sistema ou quão bem essas capacidades devem ser realizadas.

Nesta fase são capturadas as necessidades do usuário e o comportamento do sistema através da análise de casos de uso, também conhecido por "*Use Case*". As entidades externas ao sistema, em UML chamados de "atores externos", que interagem com o sistema são modelados entre as funções que eles requerem.

O diagrama de caso de uso existe dentro de um contexto estrutural que consiste de um sistema juntamente com os atores associados a esse sistema. É usado para identificar como o sistema se comporta em várias situações que podem ocorrer durante sua operação. Os componentes principais deste diagrama são os atores e os casos de uso, como mostra a Figura 3.

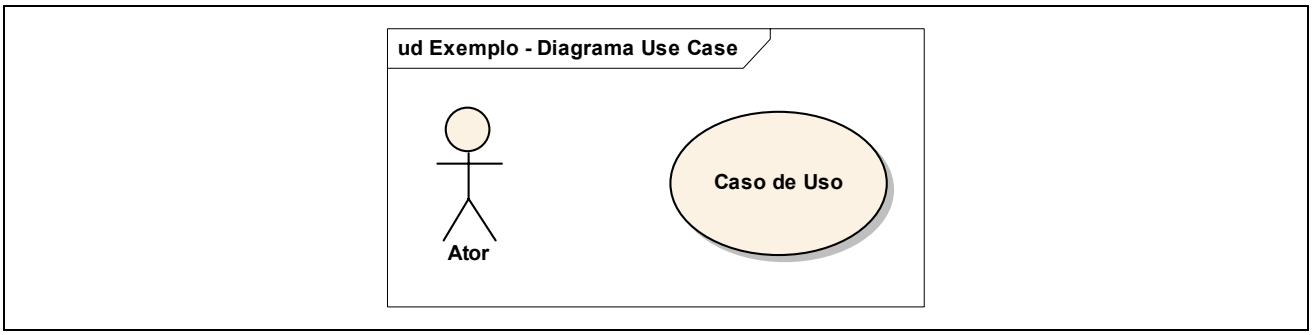

Figura 3. Notação usada pelo diagrama de caso de uso

Um ator é um objeto que está fora do escopo do sistema, mas que atua de forma significante no mesmo, podendo ser uma pessoa, um sistema, um dispositivo de entrada de dados etc. Cada ator participa de um ou mais casos de uso, interagindo com o caso de uso através de mensagens. O ator não é necessariamente um agente humano, podendo ser outro sistema ou algum processo executável. A implementação interna de um ator não é relevante no caso de uso e pode ser caracterizada por um conjunto de atributos que definem seu estado.

Segundo Ivar Jacobson (2000), um caso de uso especifica o comportamento de um sistema ou de parte dele, sendo uma descrição de um conjunto de seqüências de ações, incluindo variantes realizadas pelo sistema para produzir um resultado observável do valor de um ator. Casos de uso podem ser aplicados para captar o comportamento pretendido do sistema que está sendo desenvolvido, sem ser necessário especificar como esse comportamento é implementando.

Os sistemas de tempo-real interagem com o ambiente externo. O conjunto de objetos externos significantes e suas interações com o sistema forma a base da análise de requisitos do

sistema. Essa interação é mostrada através do modelo de caso de uso. O caso de uso é a nomeação da habilidade de uma entidade estrutural dentro do modelo.

O caso de uso existe dentro de um contexto estrutural. No caso do sistema, esse contexto consiste no sistema e nos atores associados. Um ator é um objeto fora do escopo do sistema em discussão, mas tem interações significantes com ele. Um ator, apresentado na Figura 4, é qualquer objeto que interage diretamente com o sistema.

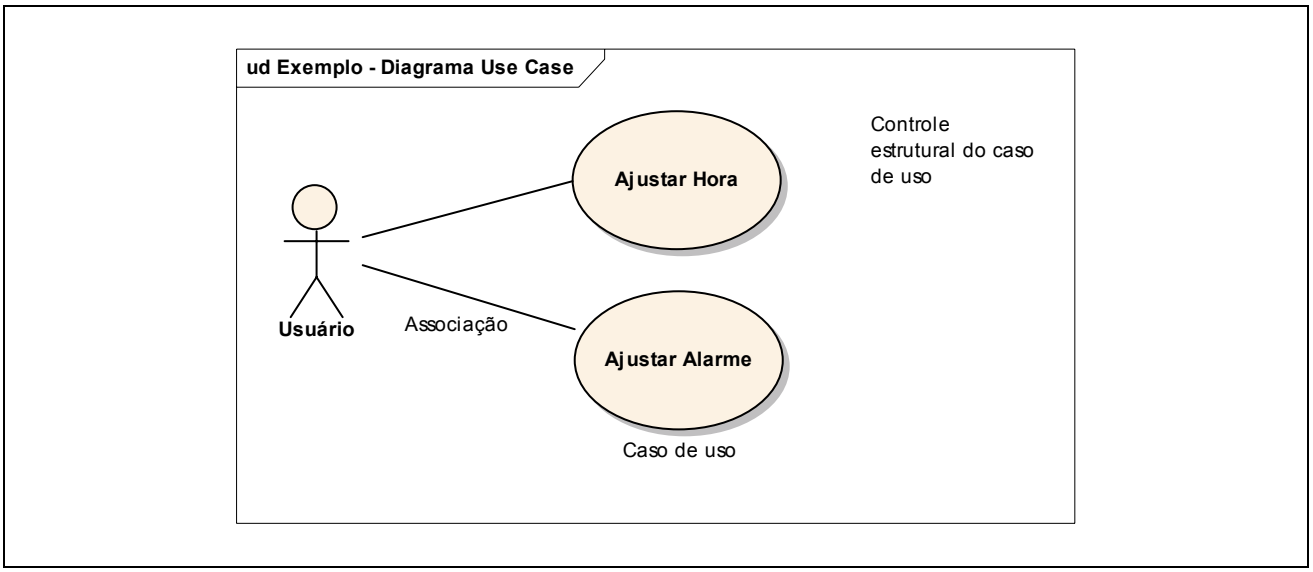

Figura 4. Exemplo de um caso de uso
Fonte: Adaptado de Douglass (2000).

Um caso de uso define um requisito funcional do sistema que é descrito em uma seqüência de passos, incluindo ações fornecidas pelo sistema e interações entre o sistema e os atores. O caso de uso mostra como os atores atuarão no sistema e descreve as ações que o sistema poderá executar.

Cada caso de uso é composto por um ou mais comportamentos. Cada comportamento é uma seqüência de passos que especifica uma ação ou interação efetuada no sistema. Cada ação especifica o processo fornecido pelo sistema e cada interação especifica as comunicações entre os sistema e os atores que participam do caso de uso.

Os casos de uso identificam as principais funcionalidades do sistema. Os detalhes dessas funcionalidades são captados através do diagramas da UML, que serão apresentados no decorrer desta seção.

2.3.1.2. Análise de Objetos

Uma vez que o ambiente externo do sistema é definido na análise de requisitos, o próximo passo é identificar os objetos chave, suas classes e suas relações dentro do sistema.

Muitas estratégias podem ser usadas para identificar objetos e classes. Esses objetos têm atributos e comportamentos que permitem que suas responsabilidades sejam desempenhadas. Classes são abstrações dos objetos e sendo assim todos os objetos instanciados de uma classe são estruturalmente idênticos.

Nesta fase são identificadas as classes, objetos e os mecanismos que solucionarão o problema. As classes são modeladas e interligadas por meio de relacionamentos utilizando o Diagrama de classes. Na análise, só serão modeladas classes que pertençam ao domínio do problema, classes que gerenciem banco de dados, comunicação, interface e outros não estarão presentes.

Um diagrama de classes mostra um conjunto de classes, interfaces e colaborações e seus relacionamentos, apresentando graficamente a visão estática do projeto de um sistema. Na maioria dos casos, isso envolve a modelagem do vocabulário do sistema, a modelagem de colaborações ou a modelagem de esquemas (JACOBSON, 2000).

A existência de uma associação entre objetos significa que um ou os dois enviam mensagens para o outro. Associações são estruturais, significando que eles podem ser parte de uma classe em que os objetos são instanciados. Instâncias das associações, chamadas de *links*, ocorrem entre objetos. Agregação e composição são formas especializadas de associação que implica num aumento nos níveis de abstração e responsabilidade.

Generalização é fundamentalmente a relação entre as classes, porque define um conjunto de atributos, comportamentos e interfaces para os descendentes das classes. Dependência significa que um elemento do modelo depende de outro de algum modo. Um exemplo comum é a dependência que a compilação tem dos pacotes.

Todos estes relacionamentos são mostrados no diagrama de classes juntamente com as suas estruturas internas, que são os atributos e operações. O diagrama de classes é considerado estático já que a estrutura descrita é sempre válida em qualquer ponto do ciclo de vida do sistema. Um sistema normalmente possui alguns diagramas de classes, já que não são todas as classes que estão inseridas em um único diagrama e uma certa classe pode participar de vários diagramas de classes.

Para suportar a colaboração entre os objetos, as classes têm relacionamentos entre si. Podem ser associações entre instâncias de classes, com associação ou agregação, ou elas podem relacionar entre as classes com generalização. Os objetos usam essas associações para se comunicarem através de mensagens.

Os diagramas de classe têm um papel muito importante em análise e projeto orientado a objeto. Eles mostram a estrutura do sistema em termos de classes e objetos, incluindo como os objetos e as classes se relacionam entre si. Um exemplo de um diagrama de classes é apresentado na Figura 5, representando a estrutura de janelas em uma aplicação simples.

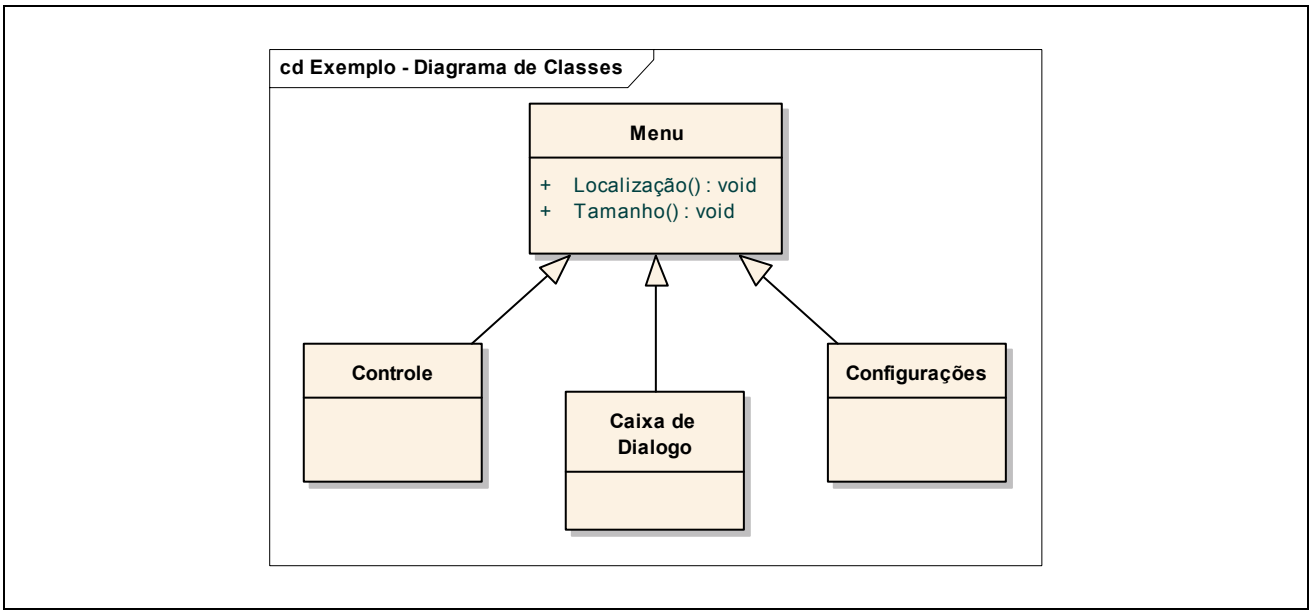

Figura 5. Exemplo de um diagrama de classe
Fonte: Adaptado de Douglass (2000).

Após a identificação dos objetos, das classes e dos relacionamentos entre eles no sistema, é feita a definição das operações e dos comportamentos dos objetos. Existem diferentes meios para a especificação do comportamento global do objeto. O comportamento liga a estrutura do objeto com seus atributos e relacionamentos, e então, pode-se determinar a responsabilidade do objeto no sistema.

O comportamento dos objetos é especificado através do diagrama de seqüência, que apresenta a seqüência de mensagens entre os objetos, facilitando a identificação dos cenários do sistema.

Um diagrama de seqüência é um diagrama de interação que dá ênfase à ordenação temporal das mensagens, é utilizado para fazer a modelagem dos aspectos dinâmicos do sistema., envolvendo a modelagem de instâncias concretas ou prototípicas de classes, interfaces, componentes e nós, juntamente com as mensagens que são trocadas entre eles, tudo isso no contexto de um cenário que ilustra um comportamento (JACOBSON, 2000).

O diagrama de seqüência mostra a interação entre os objetos ao longo do tempo. O mais importante aspecto deste diagrama é que a partir dele percebe-se a seqüência de mensagens trocadas entre os objetos. Ele mostra a interação entre os objetos, ou seja, alguma coisa que acontecerá em um ponto específico da execução do sistema.

O diagrama de seqüência consiste em um número de objetos mostrados em linhas verticais. O decorrer do tempo é visualizado observando-se o diagrama no sentido vertical de cima para baixo. As mensagens enviadas por cada objeto são simbolizadas por setas entre os objetos que se relacionam.

Na UML cada fluxo de controle independente é modelado como um objeto ativo que representa um processo ou *thread* capaz de iniciar atividade de controle. A UML define dois estereótipos padrão que se aplicam às classes ativas:

- Processo: especifica um fluxo pesado que pode ser executado concorrentemente com outros processos; e
- *Thread*: especifica um fluxo leve que pode ser executado concorrentemente com outras *threads* em um mesmo processo.

A UML utiliza a propriedade *concurrent* para dar suporte a concorrência. Em toda linguagem com suporte à concorrência pode-se elaborar um suporte para todas essas propriedades, construindo-as como semáforos, que permitirá vários leitores simultâneos do processo, mas apenas um escritor .

A Figura 6, apresenta um exemplo de diagrama de seqüência de um monitor de batimentos cardíacos, e os componentes mais comumente utilizados nesses diagramas. A linha vertical representa a instância dos objetos do sistema. As linhas horizontais representam as mensagens trocadas entre os objetos. As anotações incluídas no diagrama (apresentadas em uma folha com uma

"orelha de burro") identificam as condições iniciais, ações e atividades que não são mostradas pelas mensagens.

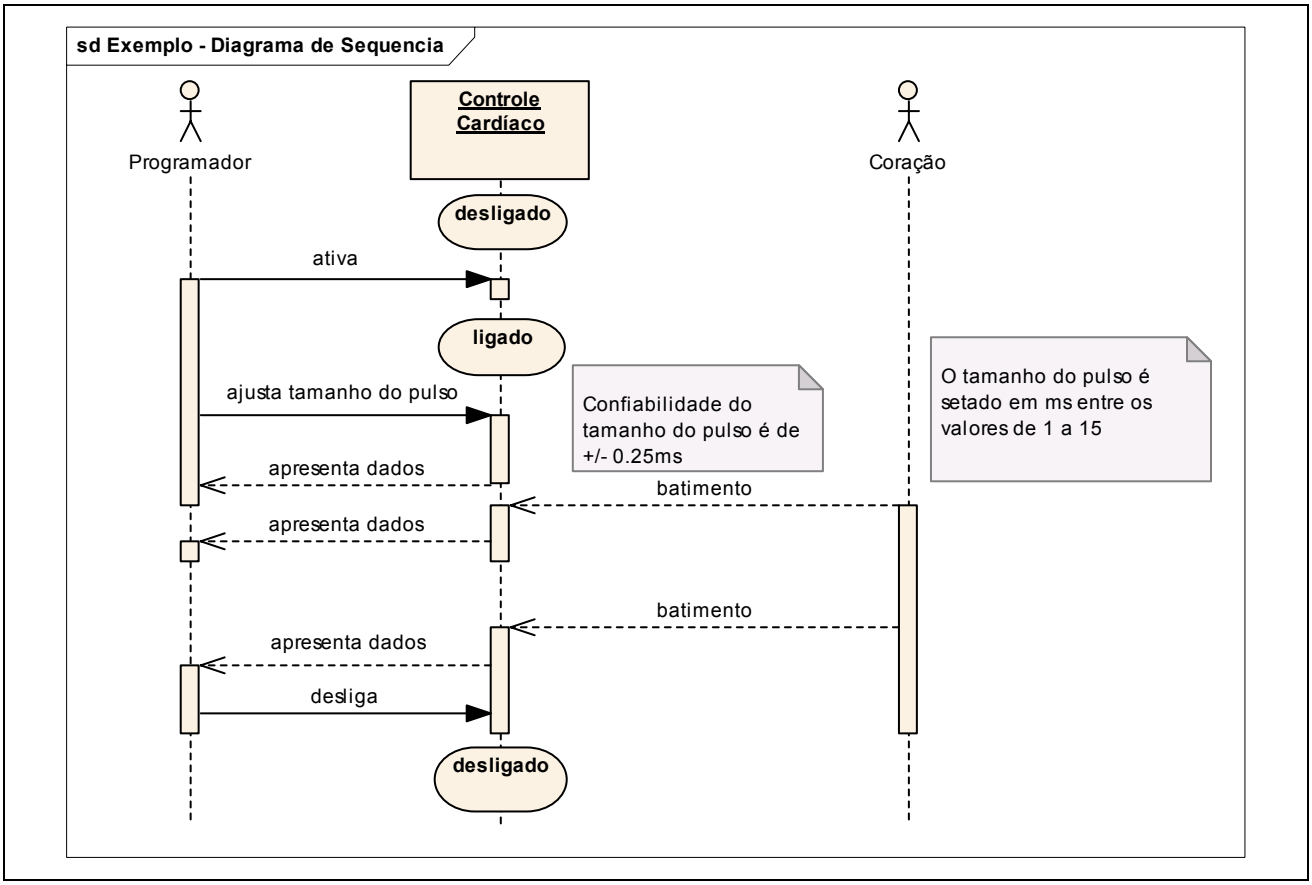

Figura 6. Exemplo de um diagrama de seqüência
Fonte: Adaptado de Douglass (2000).

2.3.2. Projeto

A análise procura especificar tudo sobre o desenvolvimento de um modelo lógico consistente e descreve as possíveis soluções aceitáveis para o problema. O projeto trata da otimização, selecionando uma solução em particular que otimiza alguns dos conjuntos de critérios do projeto, de modo que fique consistente com o modelo da análise.

Os resultado da análise é expandido nesta fase em soluções técnicas. Novas classes serão adicionadas para prover uma infra-estrutura técnica: a interface do usuário e de periféricos, gerenciamento de banco de dados, comunicação com outros sistemas, entre outros. As classes do domínio do problema modeladas na fase de análise são mescladas nessa nova infra-estrutura tornando possível alterar tanto o domínio do problema quanto a infra-estrutura. O projeto resulta no

detalhamento das especificações para a fase seguinte. Nesta fase é elaborado o diagrama de seqüência do projeto.

De acordo com Alessandra Porto (2000), os Diagramas de Componentes apresentam a arquitetura física do sistema. Os aspectos de tempo-real destes diagramas são a alocação de componentes para as classes ativas, nas quais elas serão implementadas e executadas. Uma classe componente pode ter um estereótipo «process» ou «thread», indicando a forma como uma classe ativa é implementada. A diferença importante entre processos (*process*) e tarefas (*thread*) é que um processo normalmente encapsula e protege todas suas estruturas internas através de sua execução em seu próprio espaço de memória, enquanto uma tarefa executa em um espaço de memória compartilhado com outras tarefas. Classes ativas são fisicamente construídas e distribuídas em computadores e sistemas através dos componentes nos quais elas são implementadas.

Os diagrama de componentes são empregados para a modelagem da visão estática de implementação de um sistema. Isso envolve a modelagem de itens físicos que residem em um nó, como executáveis, bibliotecas, tabelas, arquivos e documentos. Os diagramas de componentes são essencialmente diagramas de classes que focalizam os componentes de um sistema (JACOBSON, 2000).

Na Figura 7 é apresentado o diagrama de componentes e construção para um controlador de posicionamento de um telescópio. A interface com o usuário consiste em um LCD e dois botões giratórios que são controlados pelo processador.

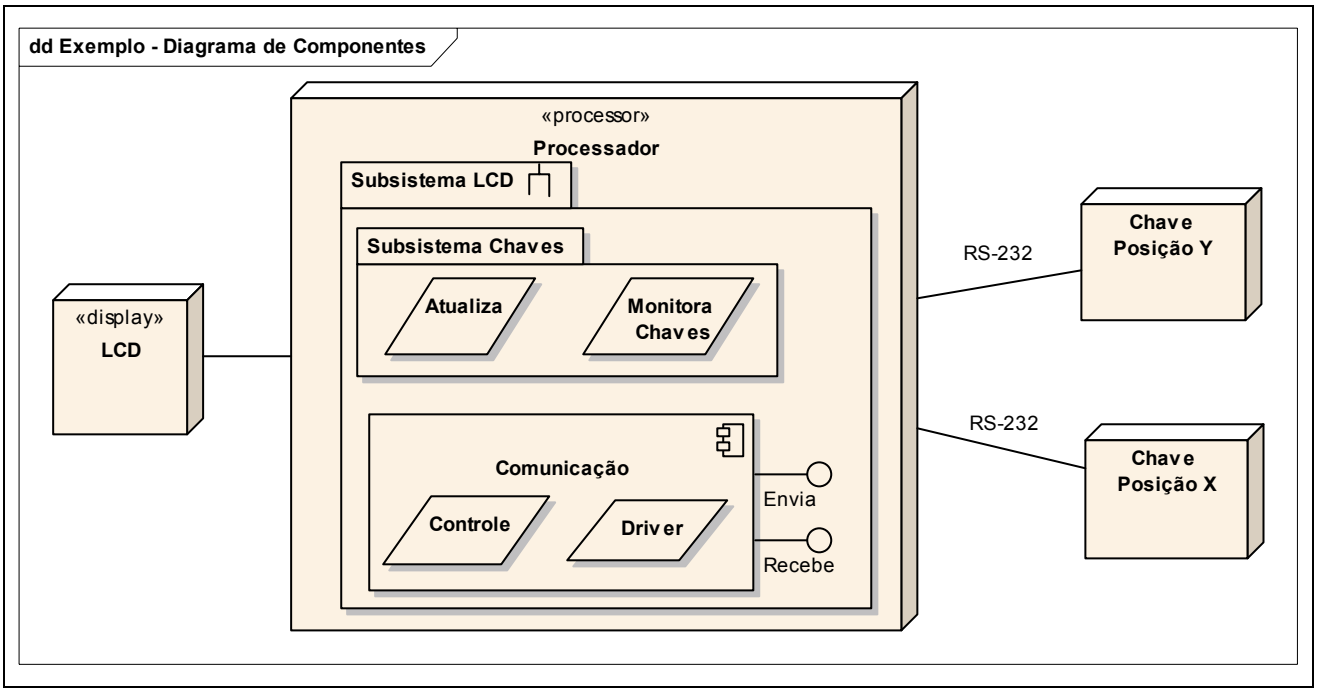

Figura 7. Exemplo de um diagrama de componentes
Fonte: Adaptado de Douglass (2000).

O processo ROPES divide o projeto em três principais categorias, como mostrado na Tabela 1, e que serão descritas nas seções a seguir.

Tabela 1. Fases de projeto

Fase de Projeto	**Escopo**	**O que é especificado**
Arquitetura	Abrangência do sistema Abrangência do processador	• Número e tipo dos processadores • Pacotes de objetos rodando em cada processador • Protocolos e meio de comunicação entre processadores • Modelo de concorrência e estratégias de comunicação entre tarefas paralelas • Camadas de software • Política de erros globais
Mecanismo	Entre objetos	• Instâncias dos padrões de projeto para múltiplos objetos • Conteúdo e nível de projeto de classes e objetos • Política de erros médios
Detalhamento	Do Objeto	• Algoritmo detalhado de cada objeto • Detalhes dos dados membros (tipos, escalas) • Detalhes da função dos membros (argumentos, estrutura interna)

Fonte: Douglass (2003).

2.3.2.1. Projeto Arquitetural

O projeto arquitetural identifica as estratégias chave para uma organização em larga escala do sistema ainda em desenvolvimento. Essas estratégias incluem o mapeamento dos pacotes de software para processadores, barramentos e seleção de protocolos, e do modelo de concorrência e tarefas paralelas.

O projeto arquitetural do sistema é mais abrangente que um simples software, e envolve a arquitetura física, como também, inclui o projeto eletrônico e mecânico. Naturalmente, a arquitetura física tem um impacto importante na arquitetura de software. Juntas, as arquiteturas física e lógica formam a arquitetura do sistema.

Classes e objetos fazem parte da arquitetura lógica do sistema, representando os conceitos lógicos e como eles interagem entre si. Os componentes fazem parte da arquitetura física. O componente é um artefato do desenvolvimento que existe quando o sistema está rodando.

Os sistemas de tempo-real normalmente possuem múltiplas tarefas paralelas de controle executando simultaneamente. Uma tarefa paralela pode ser definida como um conjunto de ações paralelas que são executadas seqüencialmente. A linguagem UML pode mostrar modelos concorrentes de diferentes modos. Os diagramas de classe e de objeto mostram as tarefas paralelas diretamente e, algumas vezes, os diagramas de estado mostram os componentes participando de múltiplas tarefas. O diagrama de seqüência mostra os cenários específicos desses objetos ativos e seus componentes interagindo com outras tarefas paralelas.

2.3.2.2. Mecanismo de projeto

O mecanismo de projeto se preocupa em adicionar e organizar as classes para suportar uma estratégia de implementação. O conjunto de classes e objetos trabalhando juntos é chamado de colaboração. A colaboração é definida de acordo com as classes específicas, rodando com regras conhecidas. Quando os colaboradores podem ser trocados por outros que preenchem essas regras, a colaboração é chamada de modelo. As regras definem a lista dos parâmetros formais para o modelo.

Os modelos padrões para o projeto são soluções para problemas com estruturas similares. O mecanismo de projeto reorganiza as entidades identificadas no projeto e adiciona objetos que facilitam a colaboração entre elas. Muitos objetos adicionados pelo mecanismo de projeto reaparecem em outros lugares, pois resolvem problemas comuns para muitos sistemas.

2.3.2.3. Detalhamento de projeto

O detalhamento de projeto adiciona informações de baixo nível necessárias para otimizar a versão final do sistema. Especifica os detalhes do projeto como o formato de armazenagem usado pelos atributos, a implementação das associações, o conjunto de operações fornecidas pelos objetos, a seleção de algoritmos internos, e a especificação de exceções dentro do objeto.

A unidade fundamental de decomposição em sistemas orientados a objetos é o objeto. O detalhamento de projeto deve considerar tanto a estrutura da informação e sua manipulação, definindo a estrutura dos dados, a implementação das associações, o conjunto de operações definidas no dado, a visibilidade do dado e suas operações, os algoritmos utilizados para implementar essas operações e as exceções do projeto.

2.3.3. Implementação

A Implementação cria o executável da aplicação a partir do modelo de projeto. A Integração usualmente não inclui apenas o código executável do projeto, mas também os objetos individuais da implementação.

As classes são convertidas para código real, preferencialmente em uma linguagem Orientada a Objetos. As linguagens procedurais não são recomendadas, a menos que não se disponha de ferramentas de desenvolvimento Orientada a Objetos para o microcontrolador utilizado no projeto. Dependendo da capacidade da linguagem usada, essa conversão pode ser uma tarefa fácil ou não.

2.3.4. Teste

A fase de teste aplica critérios de correção contra o executável da aplicação para identificar defeitos ou para demonstrar um nível mínimo de aceitação. Esta fase inclui, no mínimo, a integração e o teste de validação.

Os testes de validação são baseados no comportamento do software, segundo diversas condições que serão simuladas, para que sejam comparados com as especificações produzidas na análise de requisitos do sistema. Na maioria dos casos os testes expõem mais sintomas do que propriamente erros, ou seja, os defeitos do software não se apresentam simplesmente na interrupção do processamento ou a não-execução de uma funcionalidade, mas sim de uma forma camuflada e difícil de diagnosticar.

Inicialmente os testes são efetuados em simuladores, que reproduzem o comportamento de uma unidade de software ou hardware, apresentando os resultados o mais próximo da realidade. Como exemplo pode-se citar o Proteus-ISIS da LabCenter Electronics® utilizado nos testes das aplicações desse trabalho. O projeto é dividido em partes e a simulação do projeto é efetuada dispositivo a dispositivo.

Os simuladores substituem determinadas unidades reais que dificultam ou limitam determinados testes de software. Sua finalidade é reduzir os esforços de execução dos testes e potencializar as chances de detecção de defeitos. Proporciona flexibilidade na montagem de cenários de testes e potencializa a automação em todos os estágios de validação do software.

Os simuladores são muito úteis, pois eliminam as dependências entre o software e o hardware. Criam artificialmente cenários possibilitando testes nas mais variadas situações, disponibilizando uma infra-estrutura de testes favorável à identificação de erros.

Idêntico a qualquer outro método de modelagem. Esta fase é dividida em testes de unidades, testes de integração, teste de sistema e testes de aceitação.

2.4. METODOLOGIA WOLF

Nesta sub-seção, é apresentada a metodologia para o projeto de sistemas embarcados descrita por Wayne Wolf (2001) em seu livro “Computer as Components” (como o autor não adota nenhuma denominação para a mesma, neste texto, ela será referenciada pelo nome “metodologia WOLF”). Essa metodologia é baseada nos passos de projeto ilustrados na Figura 8.

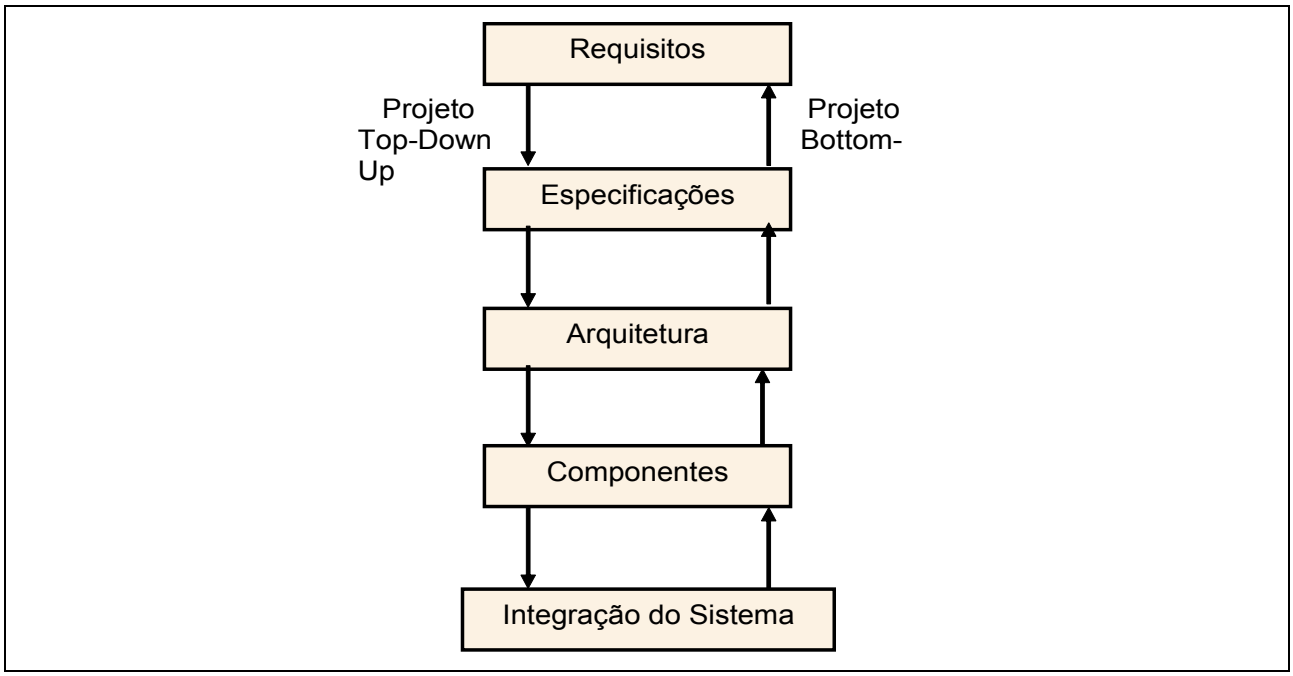

Figura 8. Passos para projeto de sistemas embarcados
Fonte: Wolf (2001).

No modelo Top-Down, o projeto inicia-se com o levantamento dos requisitos, a seguir é feita a sua especificação através de um detalhamento maior desses requisitos. Ela mostra apenas como o sistema irá se comportar e não como ele será construído. Os detalhes internos são elaborados na fase Arquitetura, que definirá a estrutura em termos de componentes. Na fase seguinte, os componentes são elaborados, incluindo os módulos do software e de hardware. Finalmente, é efetuada a implementação do projeto com base nesses componentes.

A metodologia prevê a possibilidade de utilizar um fluxo de projeto na direção oposta (Bottom-Up) nos casos em que si fizer necessário efetuar correções e/ou melhorias no projeto, voltando a fases anteriores. O projeto pode ser re-analisado a cada fase a fim de verificar se todos os requisitos estão sendo corretamente atendidos.

2.4.1. Requisitos

Antes de projetar o sistema, deve-se saber o que exatamente será projetado. As fases iniciais do processo de projeto do sistema é justamente captar essas informações para usá-las na criação da arquitetura e dos componentes do projeto. Primeiramente, descrições informais dos clientes são

adquiridas na fase de requisitos e então essas descrições são refinadas na especificação que conterá dados necessários para iniciar a fase de arquitetura do sistema.

Os requisitos são descrições informais do projeto, ou seja, descreve o que o usuário deseja. Existem dois tipos de requisitos: funcionais e não-funcionais. Um requisito funcional mostra o que o sistema deve fazer. Um requisito não-funcional pode ser qualquer número de outros atributos, incluindo o tamanho físico, custo, consumo de potência, tempo de desenvolvimento entre outros.

Vários são os requisitos que devem ser levantados nesta fase. O primeiro requisito é dar um nome coerente para o projeto, a fim de facilitar a identificação do mesmo. O propósito do sistema que identificará a função primordial do sistema. As entradas e saídas deverão ser definidas, ou seja, os tipos desses dados, as características desses dados e os tipos de dispositivos de entrada/saída.

As funções do sistema devem ser detalhadas, mostrando qual a ação tomada pelo sistema quando um novo dado chegar e quais saídas são afetadas. A performance do projeto também deverá ser analisada, pois em alguns casos, o tempo de resposta é um fator crítico e pode ocasionar alterações em todo o projeto. O custo do projeto também deve ser estipulado, pois é uma informação necessária em qualquer tipo de projeto.

O consumo de energia é um fator crítico a ser definido, visto que a grande maioria dos sistemas embarcados é projetada para trabalhar com baterias, exigindo um consumo mínimo de potência. As características físicas, como o volume e o peso também devem ser analisadas e definidas, pois em algumas aplicações, esse fator é muito importante.

2.4.2. Especificação

A especificação detalha os requisitos do sistema de forma mais precisa e com descrições consistentes que poderão ser usadas para criar a arquitetura. Nesta fase é apresentado o que o sistema faz e não como ele o faz.

A metodogia WOLF assim como a metodologia Ropes, também utiliza alguns artefatos da UML para a confecção de alguns diagramas pertinentes ao sistema em desenvolvimento. Como esses conceitos já foram apresentados previamente, eles não serão detalhados nesta seção.

O diagrama de classes e o diagrama de sequência, já descritos na metodologia ROPES, são utilizados nesta fase, definindo os relacionamentos e o comportamento dos objetos detectados na fase anterior.

Diferentemente da metodologia ROPES, a metodologia WOLF não utiliza apenas a UML para a especificação do sistema, ou seja, ela deixa a cargo do projetista a escolha e definição dos diagramas que serão utilizados no desenvolvimento do sistema. Como alternativa para a especificação do sistema o projetista poderá optar pela utilização da máquina de estados para a representação dos estados do sistema, ou ainda, especificar seu sistema através do diagrama de fluxo de dados.

A máquina de estados finitos (MEF) é uma técnica que auxilia na especificação de requisitos do sistema, pois na sua construção os possíveis estados que o sistema analisado poderá assumir deverão ser analisados, tendo como resultado um diagrama visual com os eventos que causam a transição de um estado para outro e as ações resultantes da mudança de estado.

Alan C. Shaw (2003) faz uma introdução às máquinas de estados finitos como sendo "notações naturais para descrever e raciocinar sobre circuitos de chaveamento e de vários algoritmos de software dirigidos por eventos" e explicando seu funcionamento relata que a máquina de estados finitos padrão,

> contém um número finito de estados e uma função de transição ou de "próximo estado" que mapeia estados e eventos (entradas) para estados. Em qualquer momento, o sistema está em um dado estado. A ocorrência de um evento possibilita ao sistema mudar de estado de acordo com a função de transição definida. Uma máquina normalmente tem um estado inicial e zero ou mais estados de parada. Uma MEF inicia executando a partir de seu estado inicial, troca de estados de com a entrada que recebe, até que atinja um estado de parada e/ou exaure a entrada. A forma gráfica de uma MEF é um diagrama de estados, o qual é um grafo rotulado dirigido em que os vértices denotam estados e os arcos representam transições.

A entrada para qualquer MEF é considerada uma seqüência ou sentença de símbolos de um conjunto finito. Os símbolos poderiam ser interpretados de várias maneiras diferentes, por exemplo, como caracteres, eventos comandos, dados de entrada ou palavras. Quando uma transição ocorre, o símbolo de entrada associado é consumido. Um exemplo de uma MEF é apresentado na Figura 9, a qual representa o comportamento de um portão que pode estar em um dos quatro estados: aberto, fechado, abrindo e fechando. Os principais eventos são *fp* e *ap*, que são os comando para fechar e

abrir o portão, respectivamente; e *a-a* e *f-f*, que indicam que, a partir da entrada do sensor, o portão abriu-se completamente e que o portão completou seu fechamento, respectivamente (SHAW, 2003).

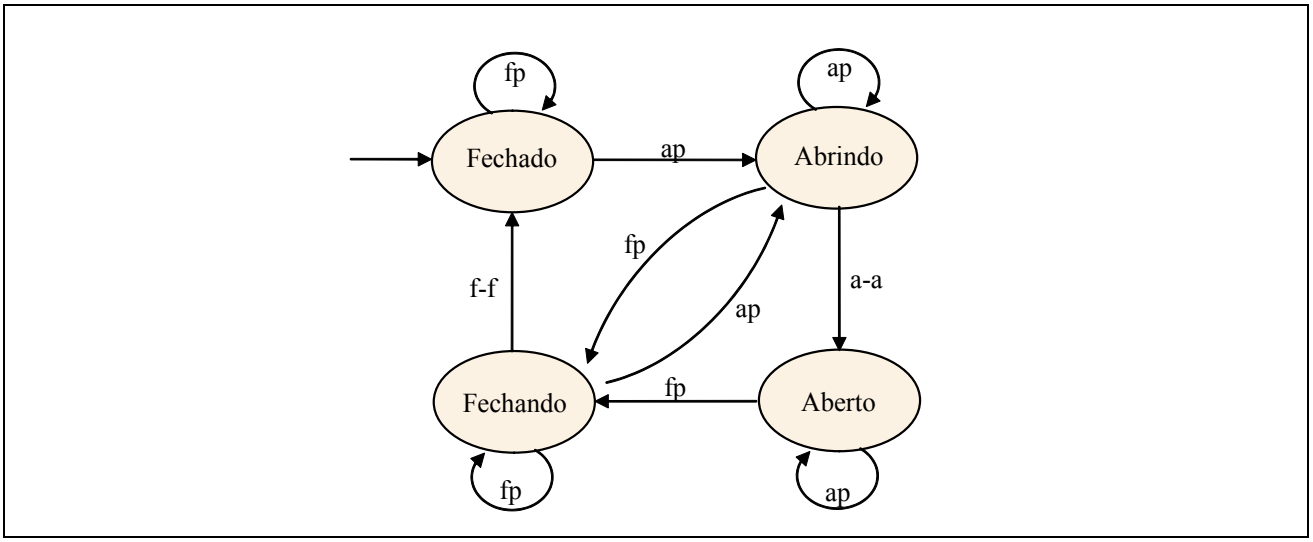

Figura 9. Exemplo de uma máquina de estados finitos para um portão
Fonte: Adaptado de Alan C. Shaw (2003)

A UML possui o diagrama de estados que em sua essência é uma máquina de estados. Nos diagramas pertinentes a representação dos estados das aplicações deste trabalho, optou-se pela notação utilizada pela UML para representar o diagrama de estados. Um exemplo de um diagrama de estados é mostrado na Figura 10.

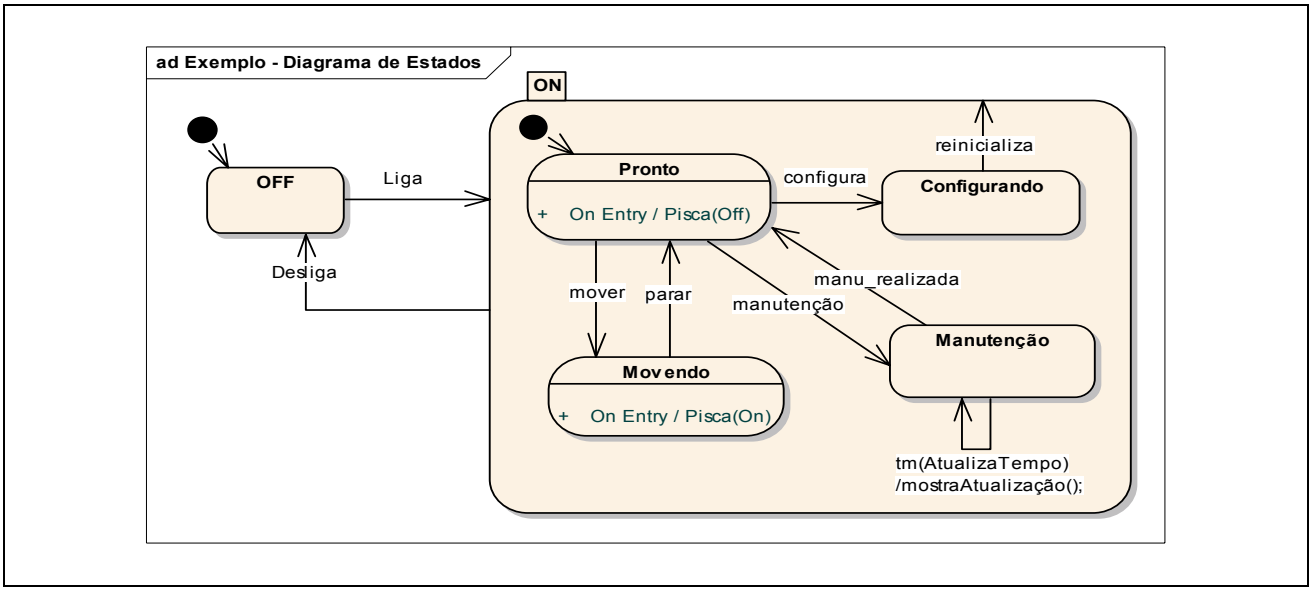

Figura 10. Exemplo de um diagrama de estados
Fonte: Adaptado de Wolf (2001).

O fluxograma do sistema também é apresentado pela metodologia WOLF como ferramenta para a especificação dos requisitos. O fluxograma inclui estados, ações, e tanto transações condicionais como não condicionais entre os estados. Na Figura 11 é apresentado um exemplo de fluxograma que representa o processo de uma ligação telefônica feita a partir de um telefone que está no gancho.

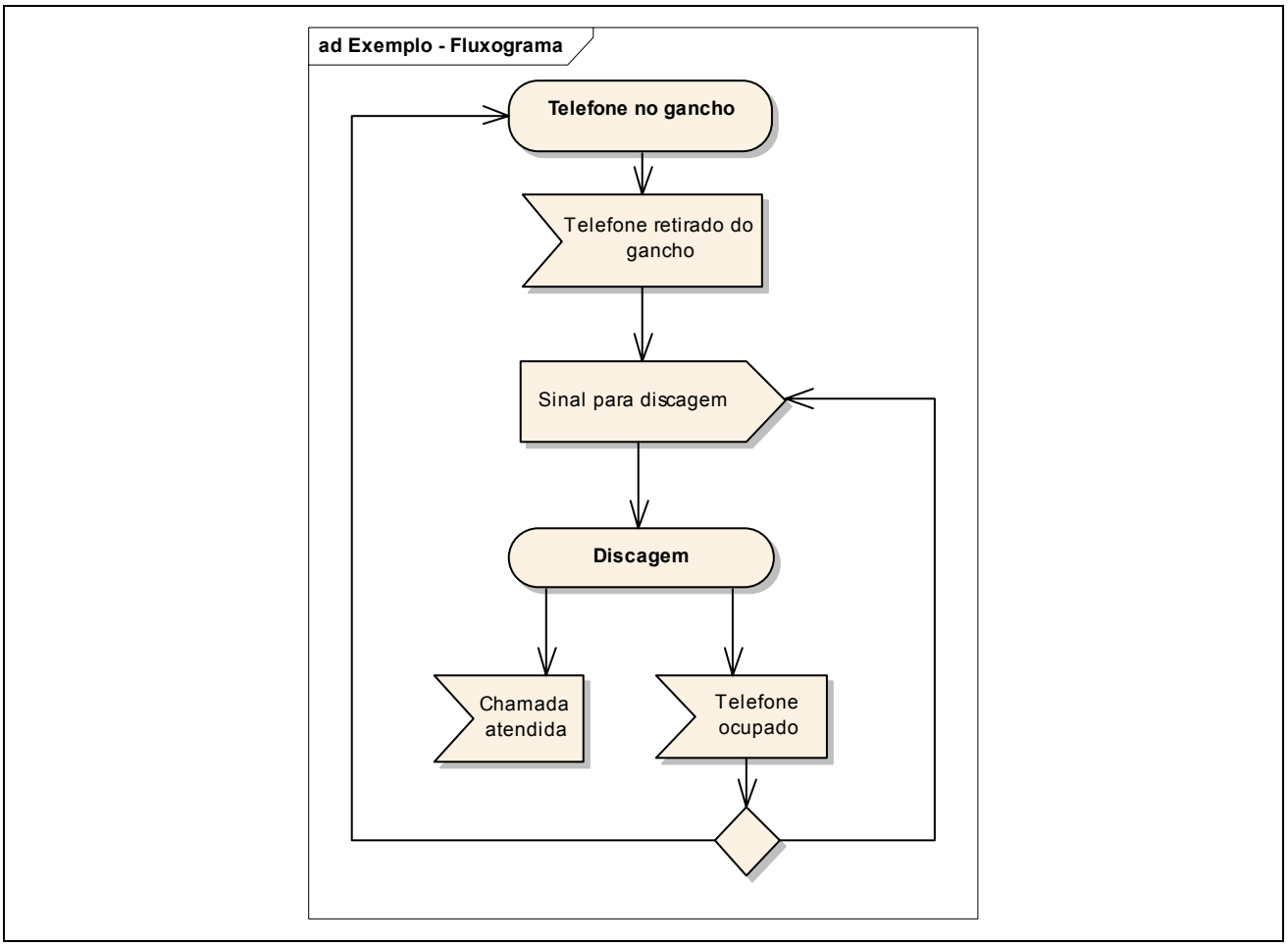

Figura 11. Exemplo de um fluxograma
Fonte: Adaptado de Wolf (2001).

2.4.3. Arquitetura

A fase de Arquitetura descreve como o sistema implementa as especificações definidas na fase anterior. Ela apresenta uma visão geral da estrutura do sistema através de Diagramas de Blocos. O Diagrama de Blocos por ser muito abstrato, não mostra quais as operações que serão realizadas pelo software do sistema, mas descreve como desenvolver as funções descritas na implementação.

Os dados levantados nesta fase serão utilizados posteriormente no projeto de componentes, pois a Arquitetura mostra quais componentes o sistema precisa. É elaborado um Diagrama de Blocos para o software e outro para o hardware do sistema.

A descrição do projeto arquitetural deve ser elaborada para satisfazer os requisitos funcionais e os não funcionais, devendo para isso apresentar tanto as funções do sistema como também mostrar os detalhes de construção, velocidade, consumo de energia, como também outras restrições e limitações.

2.4.4. Projeto dos Componentes

Na fase anterior os componentes necessários para o funcionamento do sistema foram descritos os quais serão construídos nesta fase, sendo que geralmente esses componentes incluem módulos de software e hardware.

Alguns desses componentes são padronizados, como no caso da CPU, que é um componente padrão utilizado na maioria dos casos. Muitos componentes de software podem ser reutilizados de uma biblioteca de componentes construída com projetos anteriores. O restante dos componentes serão construídos especificamente para o sistema em questão.

2.4.5. Integração do Sistema

Após a elaboração de todos os componentes, inicia-se a fase de integração. Nela os componentes são colocados para trabalhar em conjunto. Normalmente o sistema não funciona na primeira integração, apresentando erros difíceis de serem encontrados, o que torna essa fase um tanto complicada.

A dificuldade de analisar os erros deve-se, principalmente, à escassez de ferramentas e à falta de observabilidade detalhado do sistema, pois às vezes, não é possível separá-lo em partes, desde que os componentes não funcionam individualmente como operam no sistema, e também porque, em muitos sistemas embarcados, não existem vídeos ou teclados.

O recurso de simulação, assim como na metodologia ROPES, é utilizado nesta fase, sendo um facilitador no processo de integração e testes do sistema.

2.5. MEDIDAS DE SOFTWARE

Roger Pressman (1995), explica que na maioria dos empreendimentos técnicos, as medições e as medidas ajudam-nos a entender o processo técnico usado pra se desenvolver um produto, como também o próprio produto. O processo é medido, num esforço para melhorá-lo. O produto é medido, num esforço para aumentar sua qualidade.

A qualidade dos produtos de software é traduzida através de características como a correção, eficiência, confiabilidade, portabilidade ou facilidade de manutenção. A obtenção de dados quantitativos relativos a essas características é assim fundamental para introduzir melhorias no processo de desenvolvimento (ABREU,1992).

Segundo Marcus Vinícius La Rocca Macedo (2003), as medidas de software são definidas como uma grandeza que se refere ao produto ou processo e que pode ser medida direta ou indiretamente dentro do contexto do desenvolvimento do produto de software. Uma das razões principais de coletar dados sobre uma determinada medida é definir uma base de dados a ser utilizada no processo de estimativa. Além disso, a coleta de medidas durante o andamento do projeto de desenvolvimento do produto de software permite o acompanhamento deste projeto.

Segundo Fernando Brito e Abreu (1992), são vários os possíveis enquadramentos classificativos das medidas, mencionando três deles: o objeto das medidas, isto é, o âmbito da sua aplicação, o critério utilizado na sua determinação e o método de obtenção.

Quanto ao objeto, as medidas podem ser globalmente classificadas em duas categorias:

- Medidas de produto: são aplicáveis aos produtos de qualquer das fases de desenvolvimento, quantificando, por exemplo, a sua dimensão, complexidade ou qualidade (facilidade de utilização, manutenção ou outras propriedades); e
- Medidas de processo: referem-se ao processo de concepção e desenvolvimento e podem referir-se a grandezas tais como a duração do desenvolvimento, custos totais ou o nível de experiência dos programadores.

Quanto ao critério, podem ser definidas igualmente duas categorias:

- Medidas objetivas: são geralmente obtidas através de regras objetivas e bem definidas, única forma de possibilitar comparações posteriores consistentes. Na

prática isso significa que os valores a que se chega deveriam ser sempre os mesmos, independentemente do instante, condições ou indivíduo que os determina. A obtenção destas medidas é passível de automatização; e

- Medidas subjetivas: são baseadas em atributos cuja medição não pode ser feita senão de uma forma subjetiva, sendo muitas vezes derivadas de resultados de questionários e entrevistas. A complexidade, por exemplo, é por vezes classificada numa escala de "simples" / "abaixo da média" / "média" / "acima da média" / "muito grande". Embora a generalidade dos projetos possa ser facilmente classificada nestas categorias, aqueles que se situam perto das fronteiras podem ser "encaixados" na estrutura de classificação de formas distintas, dependendo da subjetividade do técnico medidor. Um outro exemplo é a quantificação da experiência de um programador.

Quanto ao método de obtenção, podem ser classificadas como:

- Medidas elementares ou primitivas: são definidas por um atributo único, como por exemplo, a duração do período de validação ou o número de dispositivos de entrada e saída de dados; e
- Medidas compostas ou computadas: são calculadas com base em outras medidas, como por exemplo, o número de falhas por milhar de linhas de código.

Fábio Bomfim (2000) define essas classes de medição de software:

- Medições orientadas ao tamanho (ou diretas): utilizam o número de linhas de código (LOC), tamanho da memória ocupada, velocidade de execução, número de erros, em um determinado período de tempo; e
- Medições orientadas à função (ou indiretas): permitem quantificar aspectos como a funcionalidade, a complexidade, a eficiência, a manutenibilidade do sistema, entre outros aspectos.

Essas duas classes de medição são descritas com mais detalhes a seguir.

2.5.1. Medida orientada ao tamanho

Marco Aurélio Cordeiro (2000), explica que

> a medida de software mais familiar é a contagem de linhas de código. Embora esta medida possa parecer simples, existe discordância sobre o que constitui uma linha de código. Para a maioria dos pesquisadores, a medida de linhas de código não deveria contar linhas de comentário e linhas em branco, uma vez que estas servem para a documentação interna do programa e não afeta a sua funcionalidade. Um outro problema é que este sistema de medidas está fortemente ligado à linguagem de programação utilizada, impossibilitando a utilização de dados históricos para projetos que não utilizam a mesma linguagem. Este tipo de medida é mais utilizado para a obtenção de informações de realização do projeto, sendo muito difícil o seu uso em estimativas.

Um conjunto de medidas de qualidade e produtividade pode ser desenvolvido com esta técnica, conforme a Tabela 2.

Tabela 2. Medidas orientadas ao tamanho.

Produtividade	**Qualidade**	**Custo**	**Documentação**
Arquitetura	Defeitos / KLOC (mil linhas de código)	$/LOC	Páginas / KLOC

Fonte: Cordeiro (2000).

Segundo Marcus Vinícius La Rocca Macedo (2003), a medida de software orientada ao tamanho, talvez seja a medida mais básica dentro do universo do desenvolvimento do produto de software, pois se trata da contagem do número de linhas do código-fonte compiladas ou interpretadas para produzir o software executável (principal produto resultante de um projeto de desenvolvimento do produto de software).

Por fim, Roger Pressman (1995) complementa as considerações apresentadas acima definindo as medidas orientadas ao tamanho como sendo medidas diretas do software e do processo por meio do qual ele é desenvolvido.

2.5.2. Medida orientada à função

Segundo Marco Aurélio Cordeiro (2000), a medida orientada à função concentra-se na funcionalidade do software. Proposta no início da década de 70 por pesquisadores da IBM, a pedido de um grupo de usuários, cujo trabalho era identificar as variáveis críticas que determinam a produtividade da programação. Descobriram que poderiam basear a avaliação de um software medindo o valor das funções executadas pelos programas, em vez de utilizar como base o volume ou a complexidade do código dos programas. Esta medida está baseada na visão externa do usuário,

sendo independente da linguagem utilizada, permitindo calcular o esforço de programação e auxiliando o usuário final a melhorar o exame e avaliação de projetos.

A idéia fundamental da análise por ponto de função consiste em medir o tamanho de qualquer produto de software baseado em termos lógicos, orientados ao usuário. A análise por ponto de função não se preocupa diretamente com a plataforma tecnológica, ferramentas de desenvolvimento e linhas de código. Ela simplesmente mede a funcionalidade entregue ao usuário final. Nesse sentido, a medida de pontos de função avalia o produto de software e mede o seu tamanho baseando-se em características funcionais bem definidas deste sistema (MACEDO, 2003).

Por fim, Roger Pressman (1995) define as medidas orientadas à função como sendo medidas indiretas do software e do processo por meio do qual ele é desenvolvido. Em vez de contar as linhas de código ela concentra-se na "funcionalidade" ou "utilidade" do programa., sendo uma das medidas mais utilizadas na atualidade.

2.6. O MODELO CMMI

O Capability Maturity Model Integration (CMMI) definido pelo Software Engineering Institute (SEI) descreve uma estrutura de trabalho que possui todos os elementos necessários para tornar um processo de desenvolvimento de software mais eficiente e controlado. Pode ser utilizado como um guia de melhoramento do processo de desenvolvimento dentro de um projeto. O CMMI baseia-se em um modelo evolutivo de maturidade, no qual as organizações partem de uma total falta de controle e gerenciamento dos processos para gradativamente adquirir novas competências, incrementando seu nível de eficiência e maturidade em relação aos diversos processos críticos envolvidos em um desenvolvimento de software (SEI, 2005).

Segundo Alexandre Bartie (2003), o modelo de processo CMMI baseia-se em cinco níveis de maturidade organizacional. Cada nível representa um estágio de maturidade dentro do processo de desenvolvimento de software. Cada organização tem seu nível individual de maturidade, refletindo seu grau de controle sobre o processo como um todo. Nenhuma empresa consegue sair do nível 1 e chegar no nível 3 sem antes passar pelo nível 2.

O CMMI de nível 2 é o nível definido como repetível onde o planejamento e o gerenciamento dos novos projetos são baseados na experiência adquirida em projetos similares anteriormente executados. O objetivo é repetir sistematicamente as melhores práticas estabelecidas

pelas várias experiências adquiridas em projetos anteriores. Um efetivo processo de gerenciamento de software deve ser praticado, documentado, garantido, treinado, medido e constantemente melhorado (SEI, 2005).

Os problemas são identificados na mesma fase em que são gerados, evitando a propagação de erros. Os requisitos de software e todos os produtos gerados durante o desenvolvimento são sistematicamente monitorados, possibilitando a evolução do tamanho e complexidade destes. Os padrões de desenvolvimento do software são definidos e a organização garante que estão sendo sistematicamente seguidos.

3. DESENVOLVIMENTO

O capítulo anterior apresentou conceitos de sistemas embarcados microcontrolados, técnicas e metodologias utilizadas para o de projeto de tais sistemas, medidas de software e CMMI. O estudo desses conceitos, proporciona a base de conhecimento que é utilizada para o desenvolvimento de três aplicações de portes diferentes.

O processo de desenvolvimento está dividido nas seguintes fases: Requisitos, Projeto, Implementação e Teste. Cada fase está dividida em duas, sendo a primeira para a metodologia ROPES e a segunda para a metodologia WOLF. A Tabela 3 mostra as quatro fases, relacionando-as com as fases correspondentes de cada metodologia.

Tabela 3. Fases de Projeto

Fase de Projeto	**Metodologia ROPES**	**Metodologia WOLF**

Requisitos	• Análise
	  ○ Requisitos
	  ○ Objetos
		• Requisitos
		• Especificação

Projeto	• Projeto
	  ○ Arquitetural
	  ○ Mecanismo
	  ○ Detalhamento
		• Arquitetura
		• Componentes

Implementação	• Implementação	• Implementação dos Componentes
Teste	• Teste	• Integração

As fases das metodologias estão encaixadas nas quatro fases desta seção, facilitando a análise da mesma e evitando repetições desnecessárias. As fases de Implementação e Teste estão descritas na mesma seção. Essas fases foram elaboradas uma única vez para cada aplicação, devido ao fato de alguns diagramas serem utilizados pelas duas metodologias.

A seguir é realizado o desenvolvimento de cada aplicação selecionada utilizando as metodologias estudadas.

3.1. APLICAÇÃO 1: TESTADOR DE DISPLAY DE 7 SEGMENTOS

A aplicação Testador de Display de 7 Segmentos, esquematizada na Figura 12, é uma aplicação básica que efetua a verificação do funcionamento dos LEDs (Light Emitter Diodes) de um display de 7 segmentos. Ela realiza a varredura em uma chave *on/off* e aciona todos os LEDs do display quando a chave for pressionada, apagando-os quando a chave for liberada.

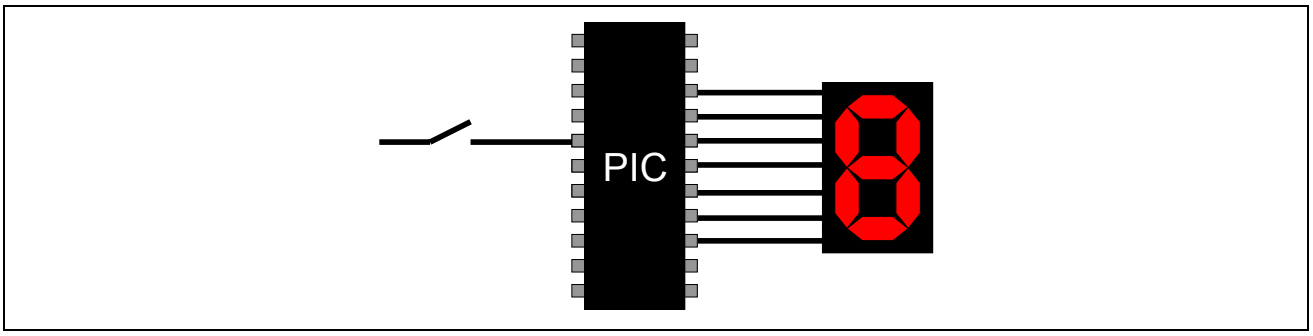

Figura 12. Aplicação Testador de Display de 7 Segmentos

3.1.1. Requisitos da Aplicação 1

3.1.1.1. Requisitos segundo ROPES

Análise de requisitos

A metodologia ROPES utiliza o diagrama de caso de uso para efetuar o levantamento dos requisitos do projeto. A aplicação Testador de Display de 7 segmentos, por ser uma aplicação básica, possui uma função bem definida, a de testar um display de 7 segmentos quando uma chave *on/off* for pressionada pelo usuário, gerando apenas um caso de uso. A Figura 13 apresenta o diagrama de caso de uso desta aplicação.

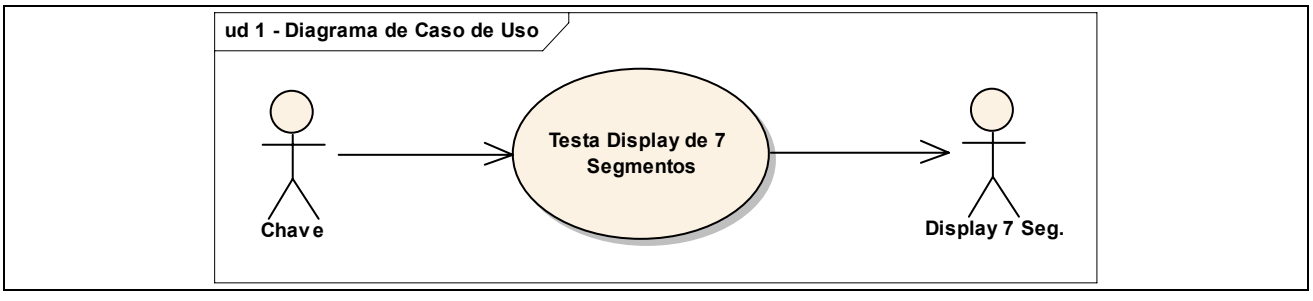

Figura 13. Diagrama de caso de uso da aplicação Testador de Display de 7 Segmentos

Análise de objetos

Na Análise de Objetos, cada objeto é detalhadamente especificado através do diagrama de classes, como mostra a Figura 14.

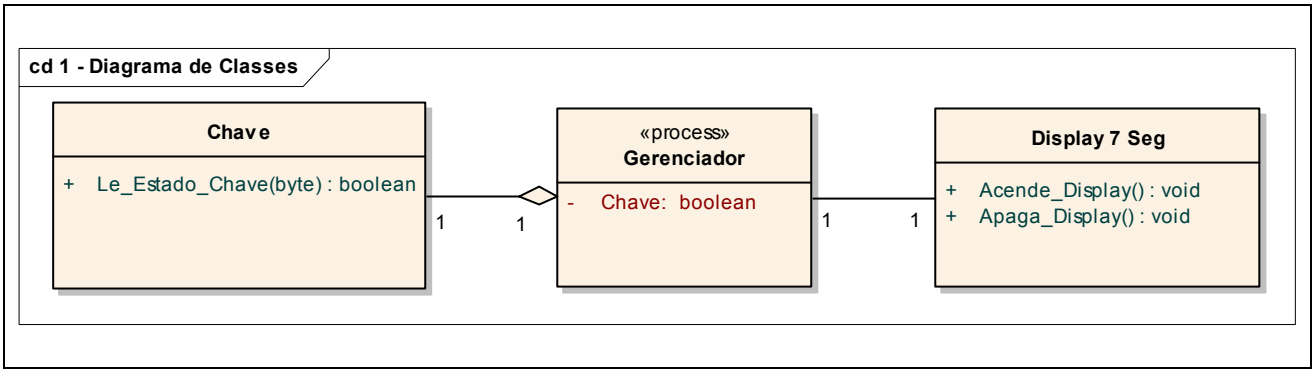

Figura 14. Diagrama de classes da aplicação Testador de Display de 7 Segmentos

A metodologia ROPES utiliza o diagrama de seqüência para o detalhamento do comportamento dos objetos do sistema. A seguir, o diagrama de seqüência da aplicação Testador de Display de 7 Segmentos é apresentado de duas formas diferentes.

O primeiro diagrama de seqüência, apresentado na Figura 15, mostra a aplicação 1 implementada por varredura, ou seja, o controle do sistema fica lendo o estado da chave *on/off* ininterruptamente e repassando essa informação para os LEDs do display de 7 segmentos.

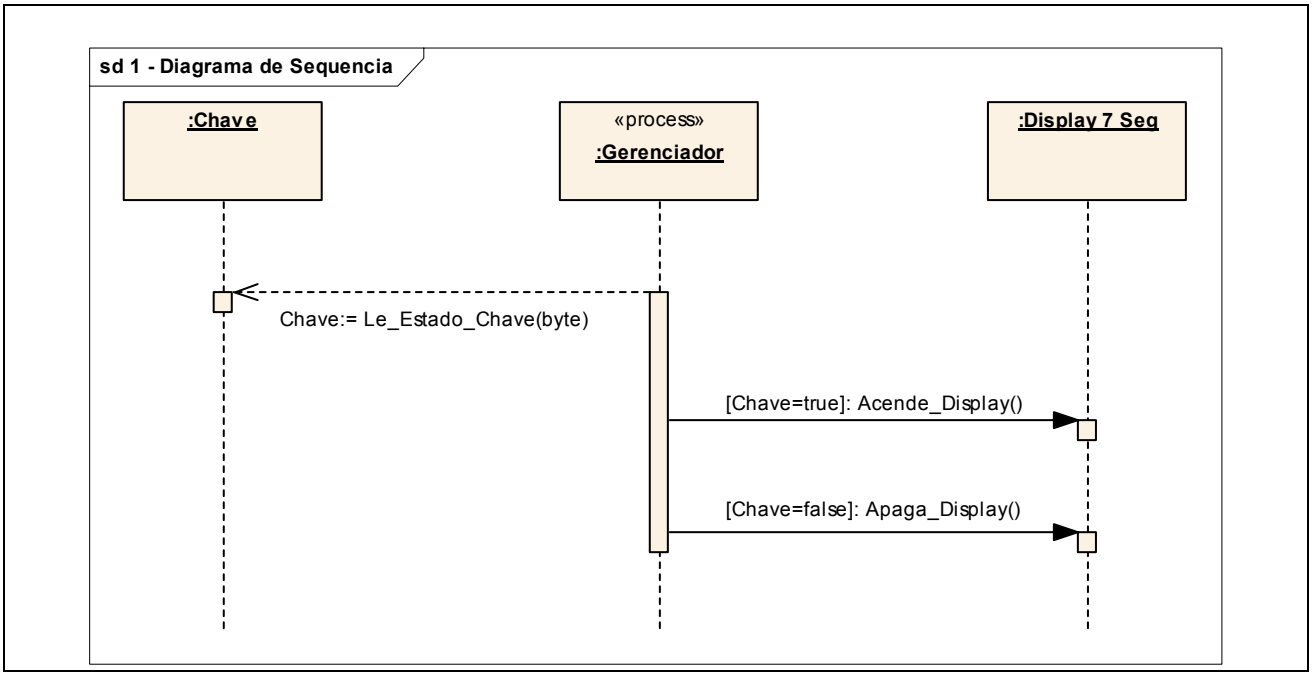

Figura 15. Diagrama de seqüência da aplicação Testador de Display de 7 Segmentos por varredura

O segundo diagrama, apresentado na Figura 16, mostra como seria a implementação se fosse elaborada por interrupção por mudança de nível, ou seja, o controle fica "adormecido" até que algo novo aconteça, que no caso dessa aplicação será a mudança de estado de uma das chaves feita pelo usuário.

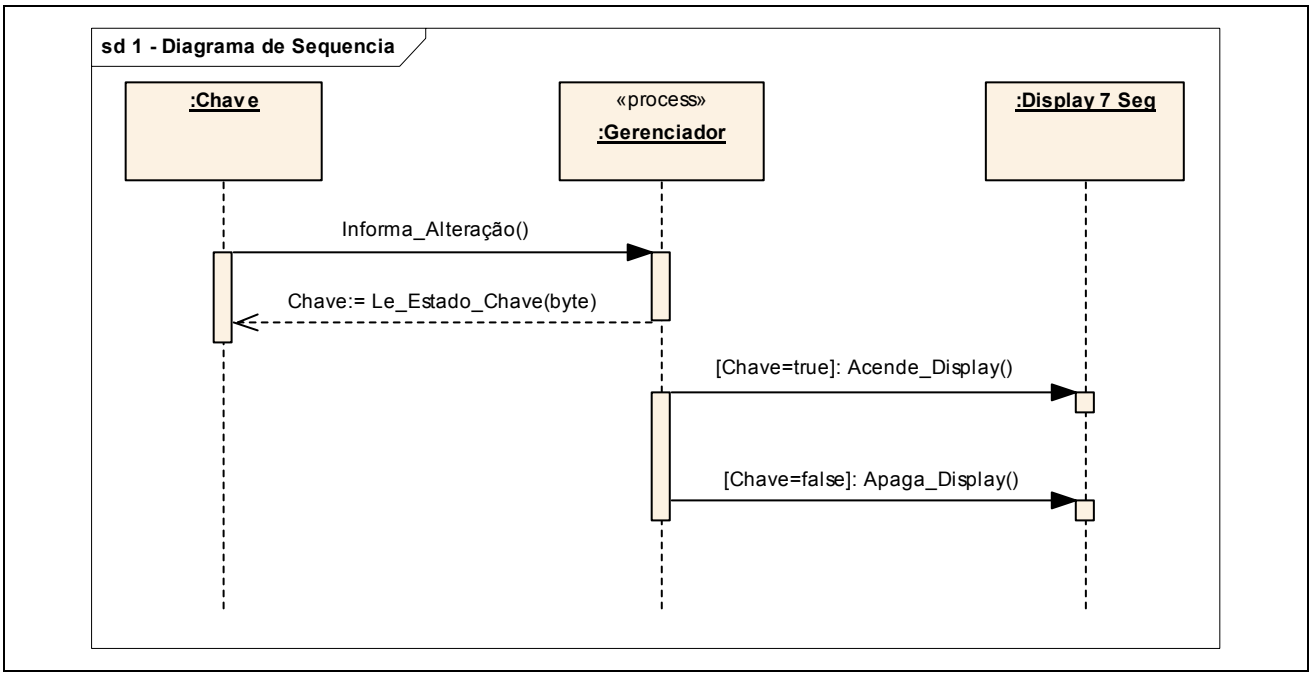

Figura 16. Diagrama de seqüência da aplicação Testador de Display de 7 Segmentos por interrupção

Neste trabalho, o modelo adotado para a implementação da Aplicação 1 foi o que utiliza varredura.

3.1.1.2. Requisitos segundo WOLF

Requisitos

Na metodologia WOLF, o levantamento de requisitos do sistema é efetuado através de uma descrição do projeto.

Os requisitos básicos desta aplicação estão descritos abaixo:

- Finalidade: realizar o teste de funcionamento de um display de 7 segmentos;
- Entrada: sinal vindo de uma chave *on/off* ligada ao sistema;
- Saídas: sete sinais para acionamento de um display de 7 segmentos; e

- Função: efetuar a varredura da chave ligada ao sistema, verificando seu estado (*on/off*) e acionando os LEDs do display de 7 segmentos.

Especificação

Nesta fase é feito um detalhamento maior dos requisitos da aplicação, como apresentados a seguir:

- Finalidade: indicar se o display de 7 segmentos está funcionando corretamente através da emissão de sinais para os LEDs do display. O sistema será implementado em microcontrolador com sua funcionalidade descrita em software;
- Entrada: sinal digital vindo da chave *on/off* ligada ao sistema. Essa chave será conectada ao sistema através de um circuito baseado em um resistor de *pull-up*. Enquanto a chave não estiver sendo pressionada (*off*), o valor lido será igual ao 1 lógico (5 V). Quando a chave for pressionada (*on*), o valor lido será igual ao 0 lógico (0 V);
- Saída: sete sinais digitais para acionamento dos LEDs do display, acionados por meio de sete pinos de uma mesma porta de entrada-e-saída; e
- Função: efetuar a leitura do sinal digital da chave *on/off* e acionar os sinais digitais que irão ligar ou desligar o display. Quando a leitura da chave for igual a 0, os pinos de saída deverão ser ativados em 1 (*on*). Quando a leitura da chave for igual a 1, os pinos de saída deverão ser ativados em 0 (*off*).

Também é elaborado o diagrama de classes, apresentado na Figura 17, para a especificação dos objetos captados pela fase anterior, e o diagrama de seqüência, apresentado na Figura 18, que define o comportamento desses objetos, já elaborados na fase de requisitos segundo ROPES.

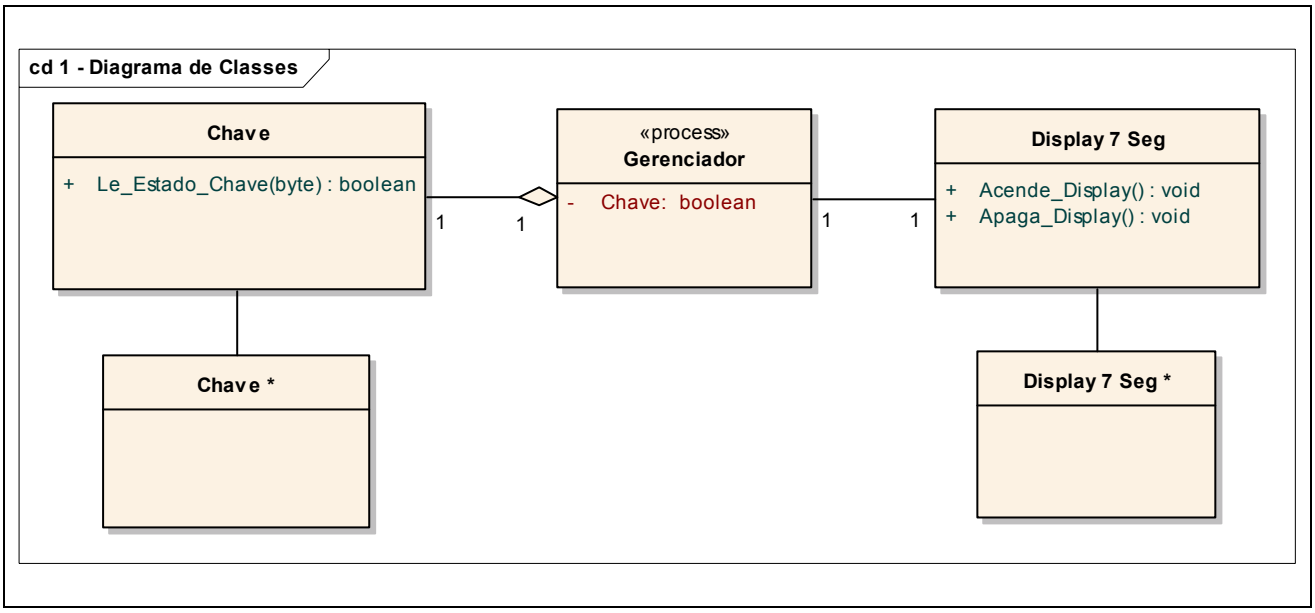

Figura 17. Diagrama de classes da aplicação Testador de Display de 7 Segmentos segundo WOLF

As classes marcadas com um '*' representam as classes de hardware da aplicação, ou seja, os tratadores do hardware correspondente.

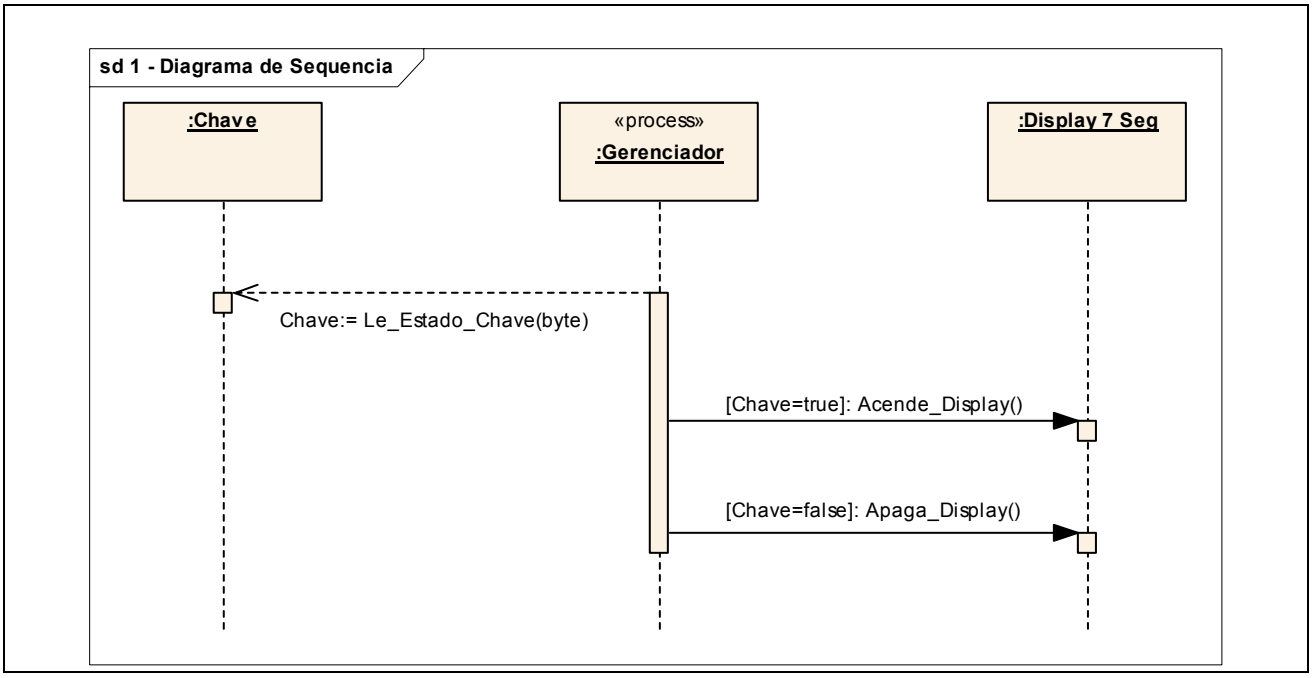

Figura 18. Diagrama de seqüência da aplicação Testador de Display de 7 Segmentos por varredura

A metodologia WOLF apresenta a máquina de estados como uma ferramenta auxiliar para a especificação do sistema. A Figura 19 apresenta a máquina de estados da aplicação Testador de Display 7 Segmentos.

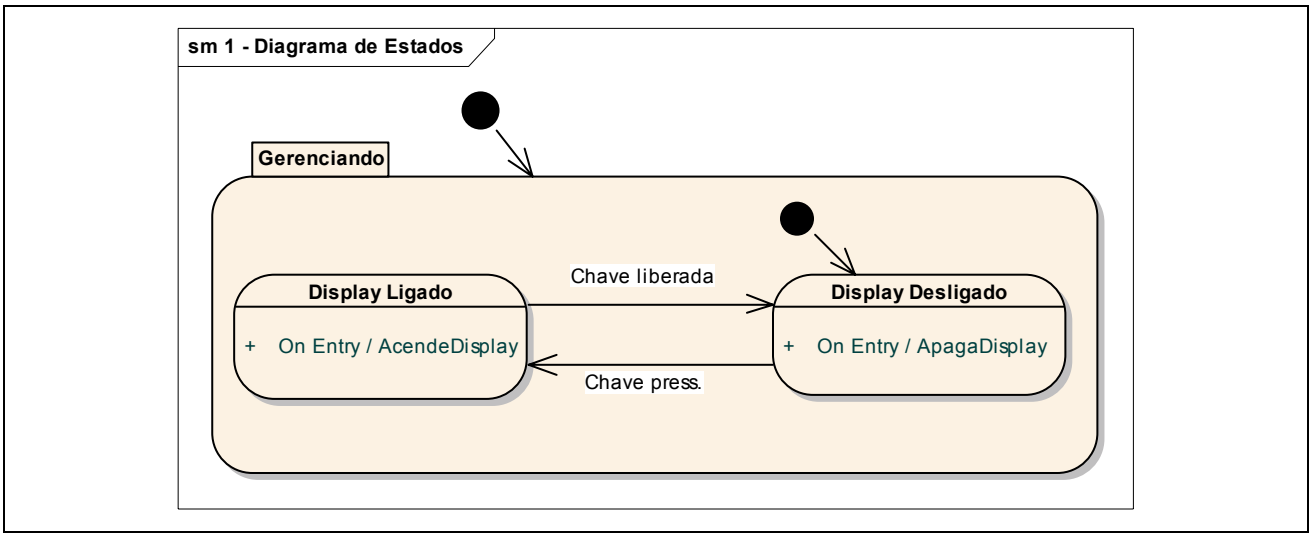

Figura 19. Diagrama de estado da aplicação Testador de Display de 7 Segmentos

Além da máquina de estados, a metodologia WOLF também coloca como opção a elaboração do fluxograma do sistema como ferramenta para documentação de especificação dos requisitos do sistema, como mostrado na Figura 20.

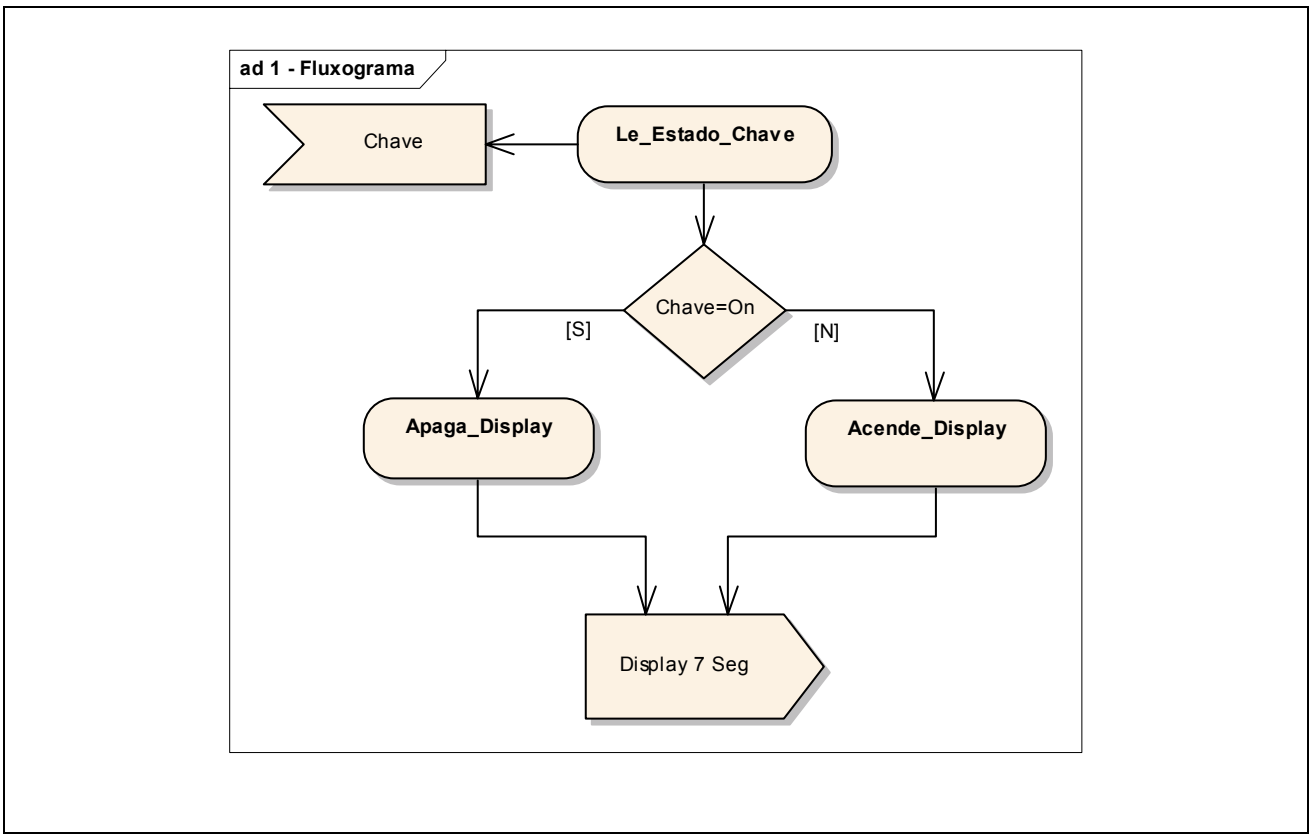

Figura 20. Fluxograma da aplicação Testador de Display de 7 Segmentos

3.1.2. Projeto da Aplicação 1

3.1.2.1. Projeto segundo ROPES

Na fase de projeto, os componentes físicos e/ou lógicos do sistema são detalhados através do diagrama de componentes e construção. Essa fase da metodologia ROPES é dividida em três: Projeto Arquitetural, Mecanismo de Projeto e Detalhamento do Projeto, sendo elaborada em um processo de refinamento sucessivo. A Figura 21 apresenta os componentes da aplicação Testador de Display de 7 segmentos.

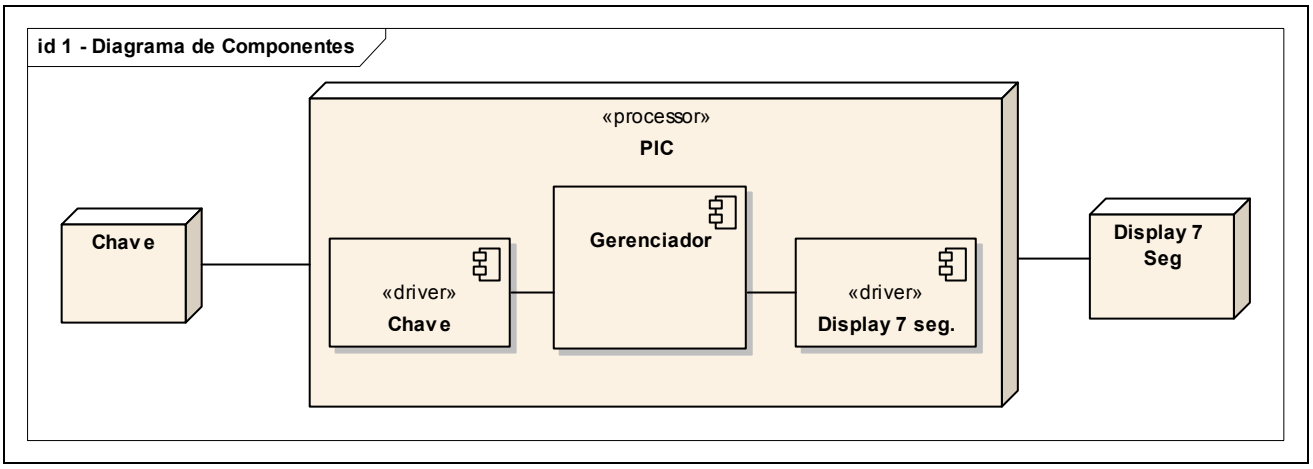

Figura 21. Diagrama de componentes e construção da aplicação Testador de Display de 7 Segmentos

3.1.2.2. Projeto segundo WOLF

A metodologia WOLF utiliza para a elaboração do projeto arquitetural os Diagramas de Blocos, tanto do software como do hardware da aplicação.

A Figura 22 apresenta o Diagrama de Blocos do Software da aplicação Testador de Display 7 de Segmentos.

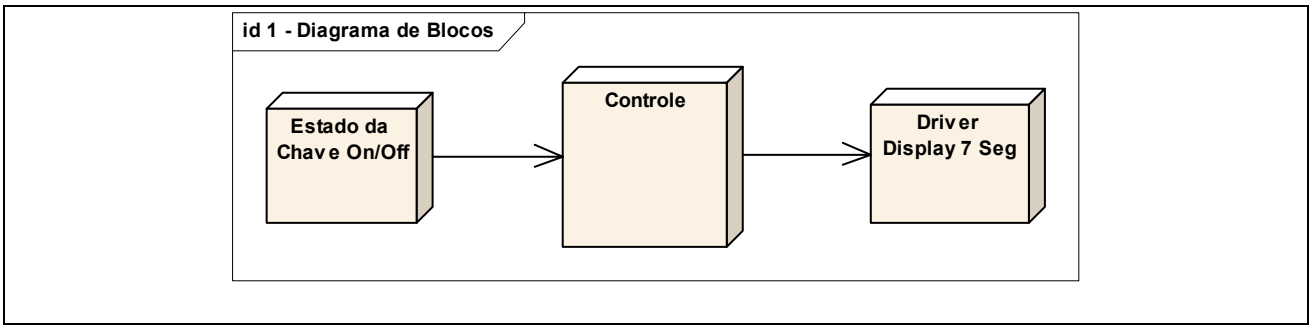

Figura 22. Diagrama de blocos do software da aplicação Testador de Display de 7 de Segmentos

A Figura 23 apresenta o Diagrama de Blocos do Hardware da aplicação Testador de Display 7 de Segmentos.

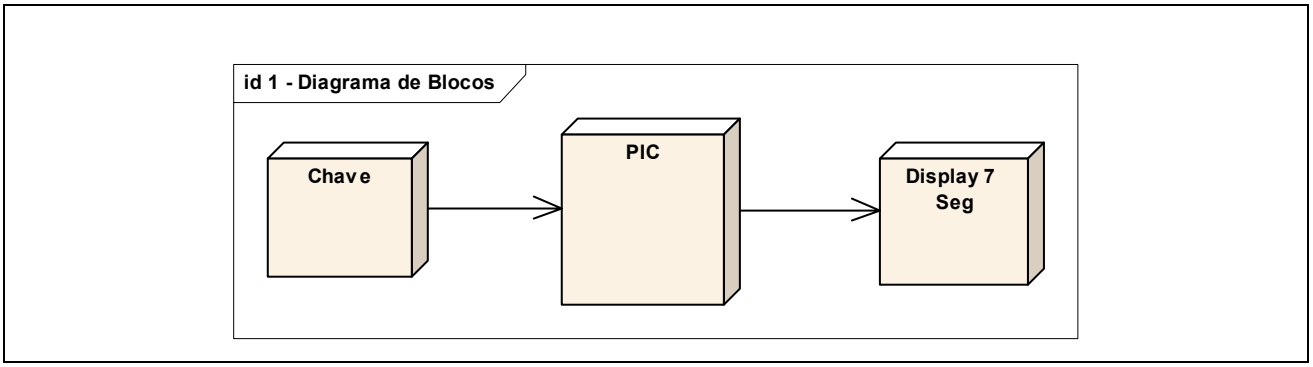

Figura 23. Diagrama de blocos do hardware da aplicação Testador de Display de 7 de Segmentos

3.1.3. Implementação e Teste da Aplicação 1

Como o foco do trabalho não envolve questões de hardware, esta fase não é descrita em detalhes, apenas apresentando o suficiente para a complementação do processo do desenvolvimento do software. Apenas uma implementação é efetuada tanto para a metodologia ROPES quanto para a metodologia WOLF.

Para a implementação do esquema de hardware, teste e validação das aplicações, foi utilizada a ferramenta chamada PROTEUS/ISIS, a qual faz simulação de circuitos e componentes eletro-eletrônicos. A Figura 24 apresenta o esquema da ligação física do hardware da aplicação Testador de Display de 7 Segmentos.

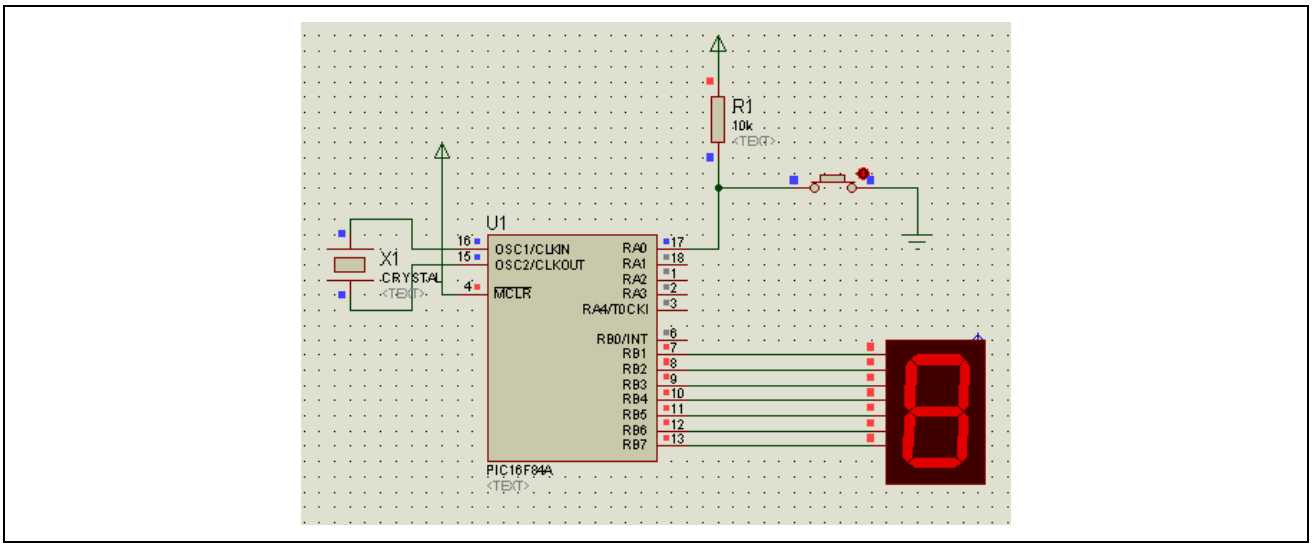

Figura 24. Esquema da ligação física do hardware da aplicação Testador de Display de 7 Segmentos

O software da aplicação Testador de Display de 7 Segmentos foi desenvolvido com a utilização do compilador C PCWH da CCS. O PCWH compiler foi desenvolvido para ambiente WINDOWS e possui a capacidade de transformar a linguagem C para a linguagem de montagem dos microcontroladores PIC.

A seguir é apresentado um trecho do código fonte da aplicação Display de 7 Segmentos.

```
(. . .)
/*******************************CHAVE*************************************/
boolean Le_Estado_Chave(byte &chave){
    return input(testa)==0;
}/*******************************DISPLAY***********************************/
void Acende_Display(){
    output_high(pin1);
    // . . .
    output_high(pin7);
}
void Apaga_Display(){
    output_low(pin1);
    // . . .
    output_low(pin7);
}/*******************************GERENCIADOR*******************************/
void main() {
   while (1) {
      delay_ms(100);
      if (Le_Estado_Chave(testa)==true){
            Acende_Display();         }
      else{
            Apaga_Display();          }
   }}
```

Figura 25. Trecho do código da Aplicação Testador de Display de 7 Segmentos

A software da aplicação Testador de Display de 7 Segmentos, foi testado e validado na ferramenta de simulação PROTEUS/ISIS, apresentado na Figura 24, e no Kit de prototipação disponível no LSED.

3.2. APLICAÇÃO 2: CRONÔMETRO

A aplicação Cronômetro, esquematizada na Figura 26, também é uma aplicação básica que efetua uma contagem de tempo progressiva, com a resolução de centésimos de segundos, atingindo um valor máximo de 59’59’’99. Ela realiza a varredura em duas chaves (*push-button*): S1 e S2. Quando a chave S1 for pressionada, a contagem deverá ser paralisada e o valor de contagem zerado (reset). Quando a chave S2 for pressionada pela primeira vez, a contagem deverá ser iniciada. Quando a chave S2 for pressionada uma segunda vez, a contagem deverá ser paralisada. Se for pressionada novamente, a contagem deverá ser retomada com o valor que estava quando foi paralisada. A contagem deverá ser apresentada ao usuário em um display de cristal líquido (LCD).

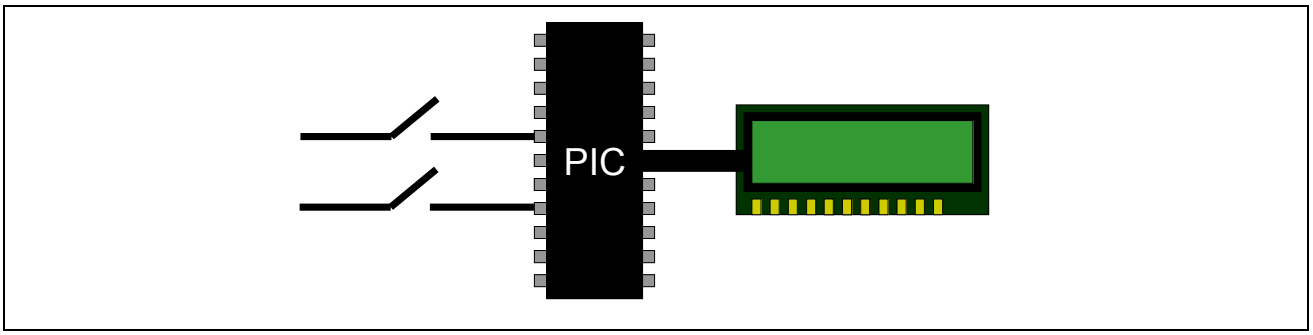

Figura 26. Aplicação Cronômetro

3.2.1. Requisitos da Aplicação 2

3.2.1.1. Requisitos segundo ROPES

Análise de requisitos

Para facilitar a visualização das funções da aplicação, o caso de uso da aplicação Cronômetro, foi dividido em dois níveis de abstração. A Figura 27 apresenta o caso de uso geral dessa aplicação.

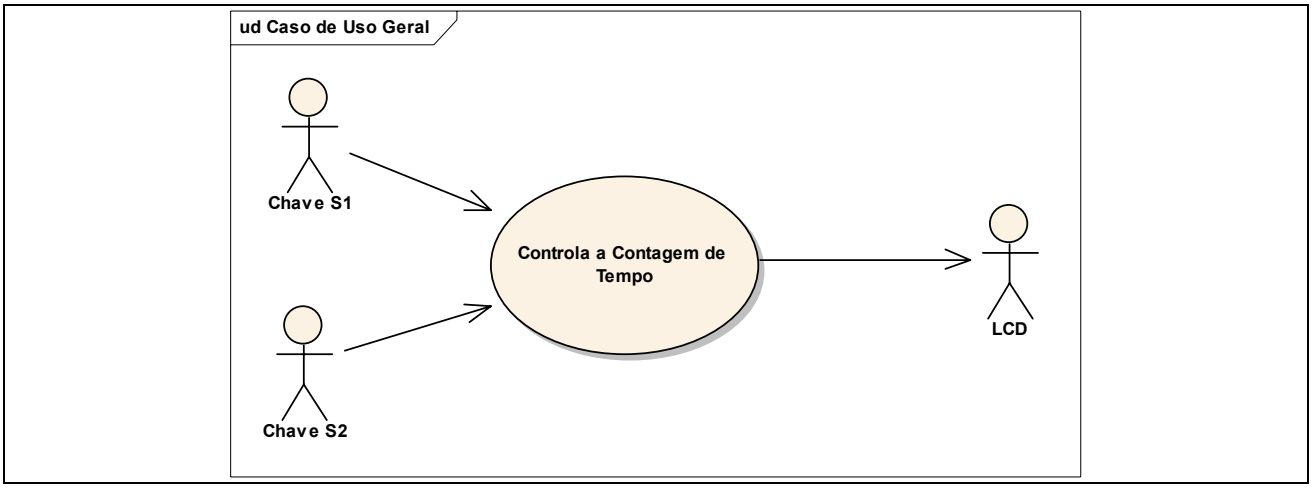

Figura 27. Diagrama de caso de uso geral da aplicação Cronômetro

A Figura 28 apresenta o detalhamento do caso de uso da aplicação Cronômetro. O primeiro caso de uso, verifica se as chaves foram pressionadas por um tempo mínimo (0.5 ms), passando essa informação para o próximo caso de uso que realiza a contagem de tempo de acordo com o estado do sistema e das chaves de entrada, descrevendo o seguinte comportamento:

- Chave S1 pressionada: paralisa a contagem e zera o contador;
- Chave S2 pressionada:
  - 1ª vez: inicia a contagem
  - 2ª vez: paralisa a contagem
  - 3ª vez: retoma a contagem

E o ultimo caso de uso, completa a descrição das funcionalidades dessa aplicação, acionando o LCD para exibir o estado atual da contagem.

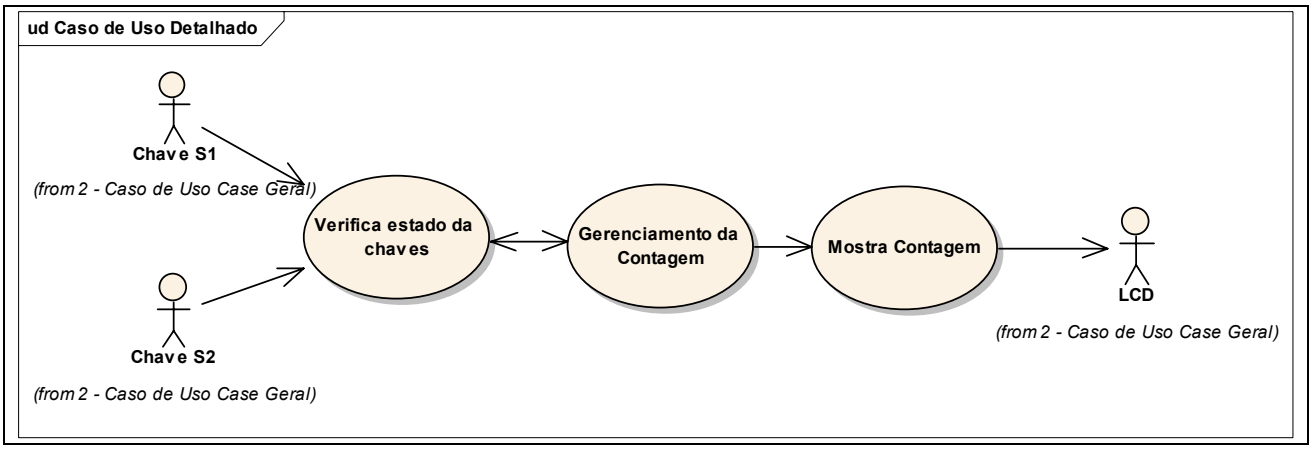
ud Caso de Uso Detalhado
Chave S1
(from 2 - Caso de Uso Case Geral)
Chave S2
(from 2 - Caso de Uso Case Geral)
Verifica estado da chaves
Gerenciamento da Contagem
Mostra Contagem
LCD
(from 2 - Caso de Uso Case Geral)

Figura 28. Diagrama de caso de uso detalhado da aplicação Cronômetro

Análise de objetos

O detalhamento dos objetos da Aplicação 2 é apresentado na Figura 29, através do diagrama de classes.

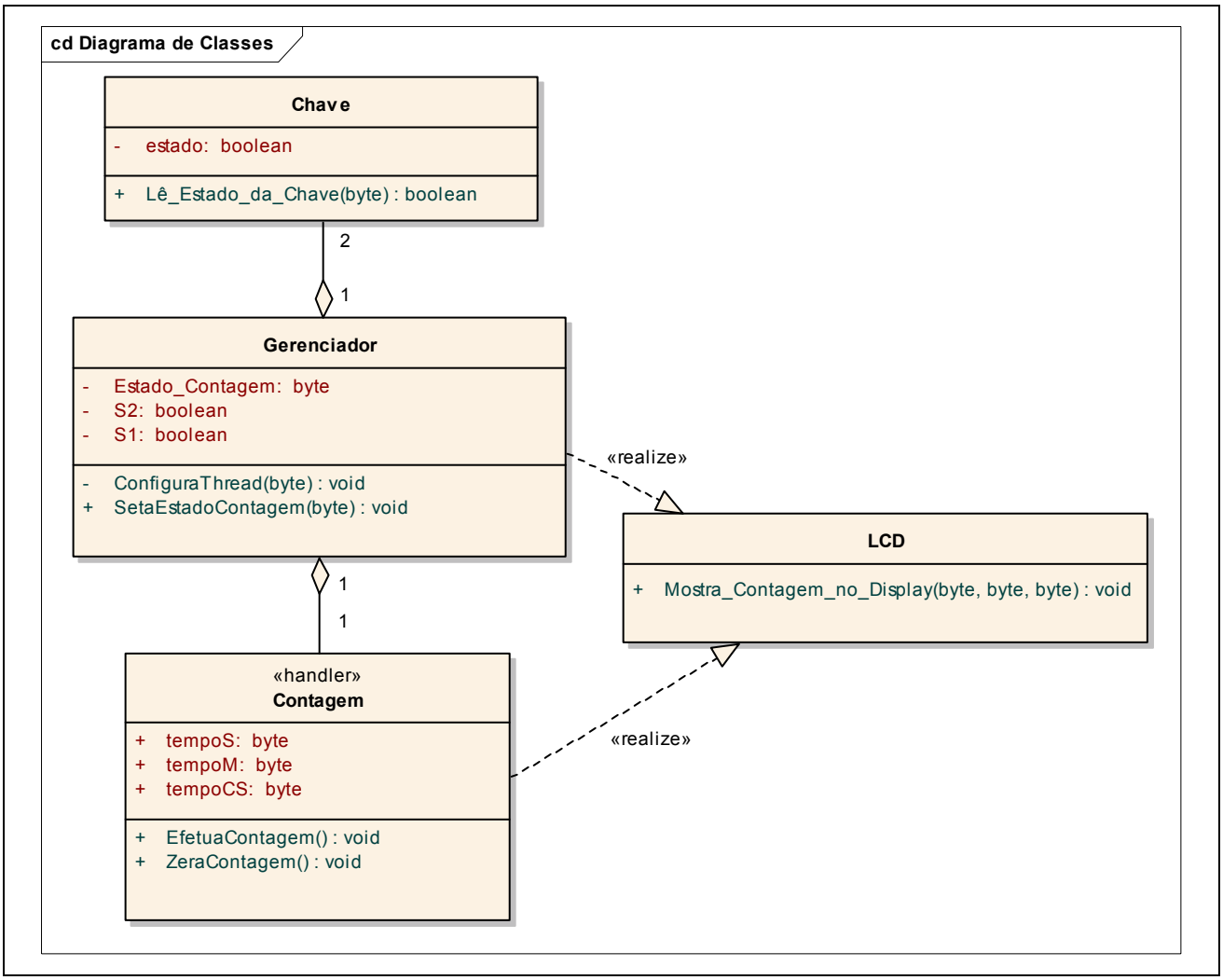

Figura 29. Diagrama de classes da aplicação Cronômetro

O diagrama de sequência da aplicação Cronômetro é apresentado na Figura 30. Como na Aplicação 1, o modelo de varredura será utilizado para essa aplicação. O controle fica varrendo as chaves e conforme seu estado executa os procedimentos pertinentes a mesma, incrementando, pausando ou zerando e parando a contagem .

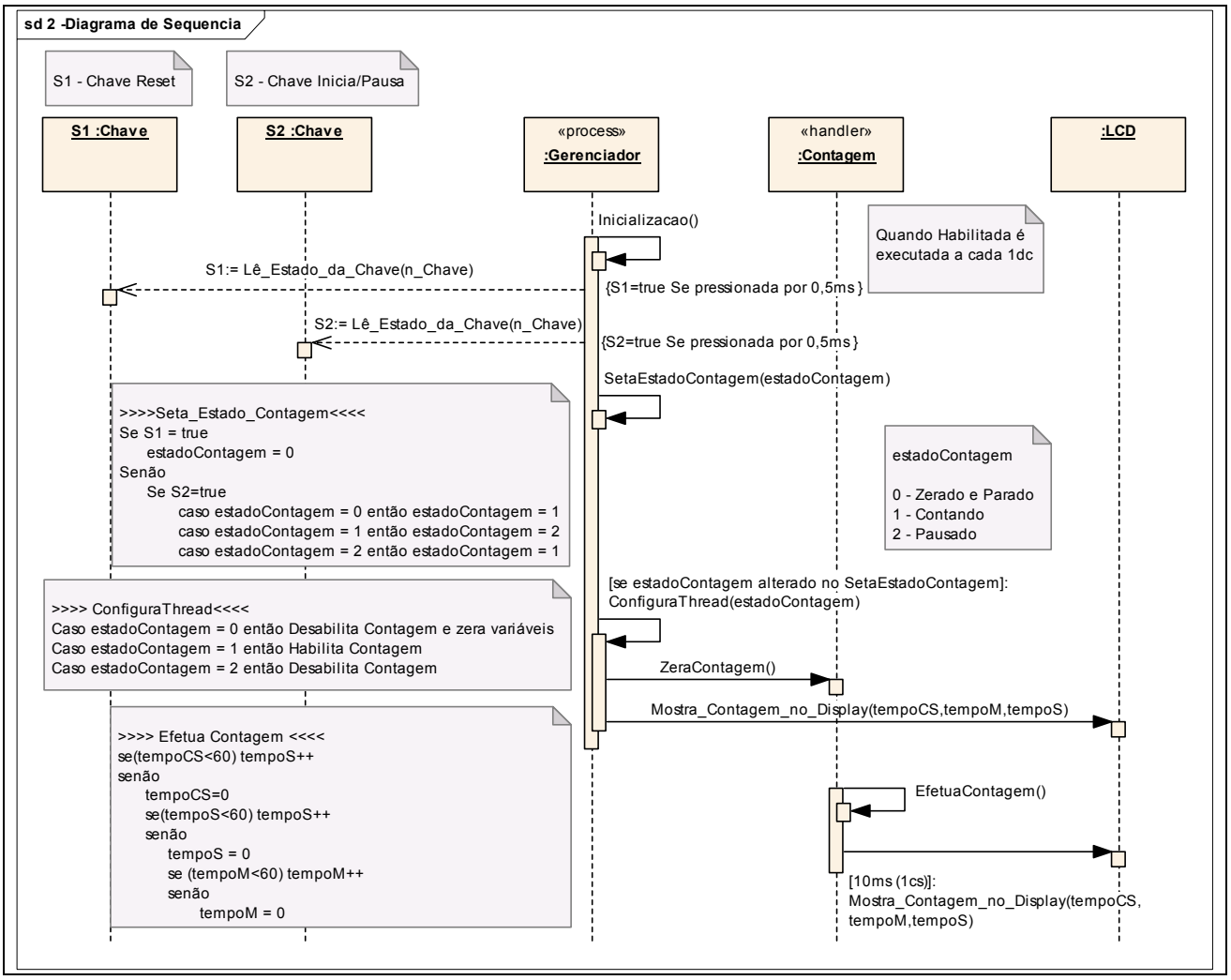

Figura 30. Diagrama de seqüência da aplicação Cronômetro

3.2.1.2. Requisitos segundo WOLF

Requisitos

Os requisitos básicos desta aplicação estão descritos abaixo:

- Finalidade: contar, pausar ou zerar e parar uma contagem de tempo quando solicitado;
- Entrada: sinais vindos de duas chaves *push-button* ligadas ao sistema;

- Saídas: sinais para acionamento de um display numérico;
- Função: efetuar a varredura das chaves ligadas ao sistema, verificando seu estado (*press/released*) atualizando o display numérico.

Especificação

Nesta fase é feito um detalhamento maior dos requisitos da aplicação, como apresentados a seguir:

- Finalidade: realizar a contagem de tempo de acordo com o pressionamento das duas chaves ligadas ao sistema, acionando, pausando ou zerando e parando a contagem de tempo e atualizando essa contagem no display numérico.
- Entrada: sinais digitais vindos de duas chaves *push-button* ligadas ao sistema. Essas chaves são conectadas ao sistema através de um circuito baseado em um resistor de *pull-up*. Enquanto a chave não estiver sendo pressionada (*off*), o valor lido será igual ao 1 lógico (5 V). Quando a chave for pressionada (*on*), o valor lido será igual ao 0 lógico (0 V);
- Saída: sinais digitais para acionamento do display numérico, que indicará a contagem de tempo para o usuário;
- Função: efetuar uma varredura nas duas chaves ligadas ao sistema, verificando seu estado (*pressed/released*) e executando a ação correspondente a cada chave;

A Figura 31 apresenta o diagramas de classes elaborado pela metodologia WOLF.

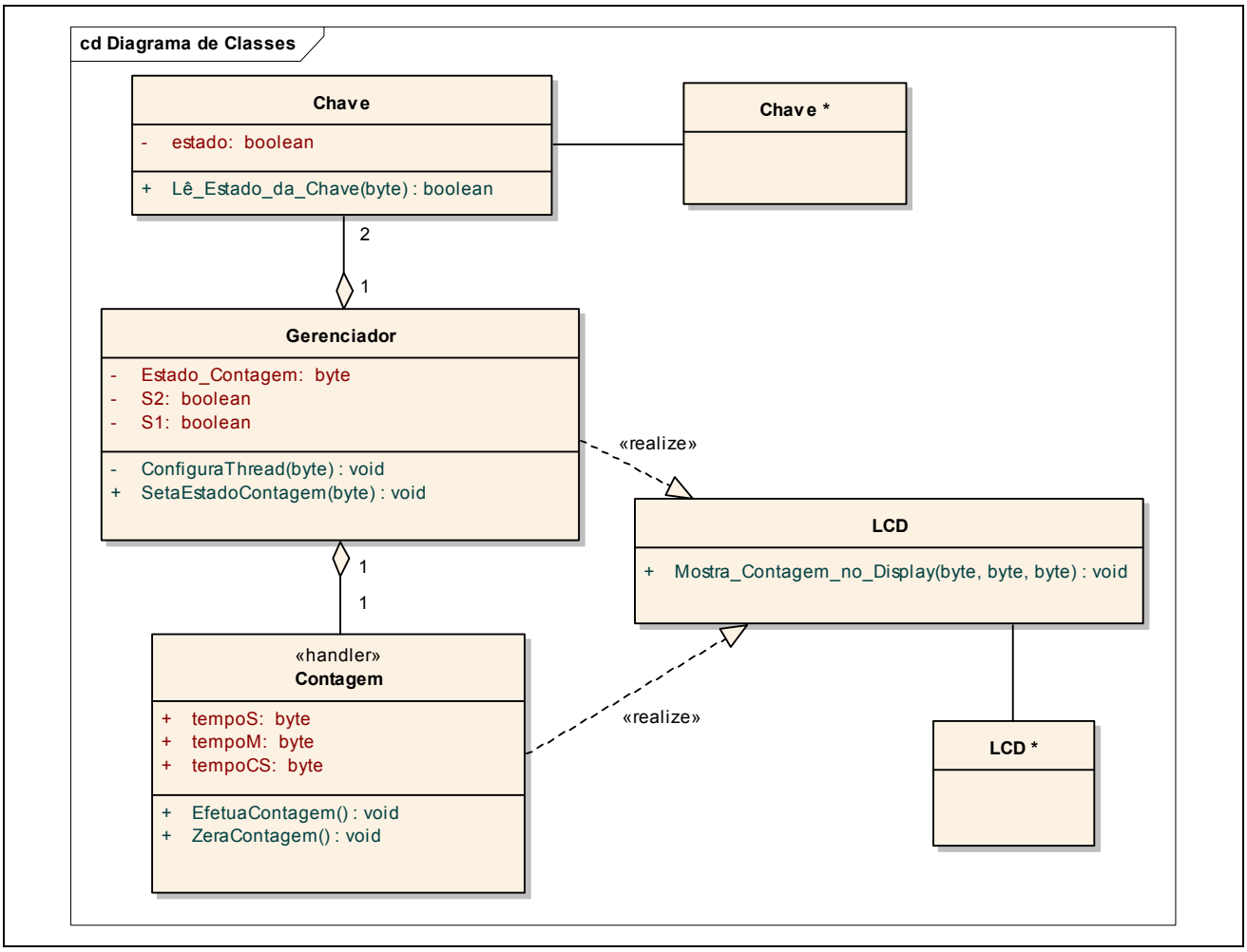

Figura 31. Diagrama de classes da aplicação Cronômetro segundo WOLF

O diagrama de sequência da aplicação Cronômetro é apresentado na Figura 32. O controle fica varrendo as chaves e conforme seu estado executa os procedimentos pertinentes a mesma, incrementando, pausando ou zerando e parando a contagem .

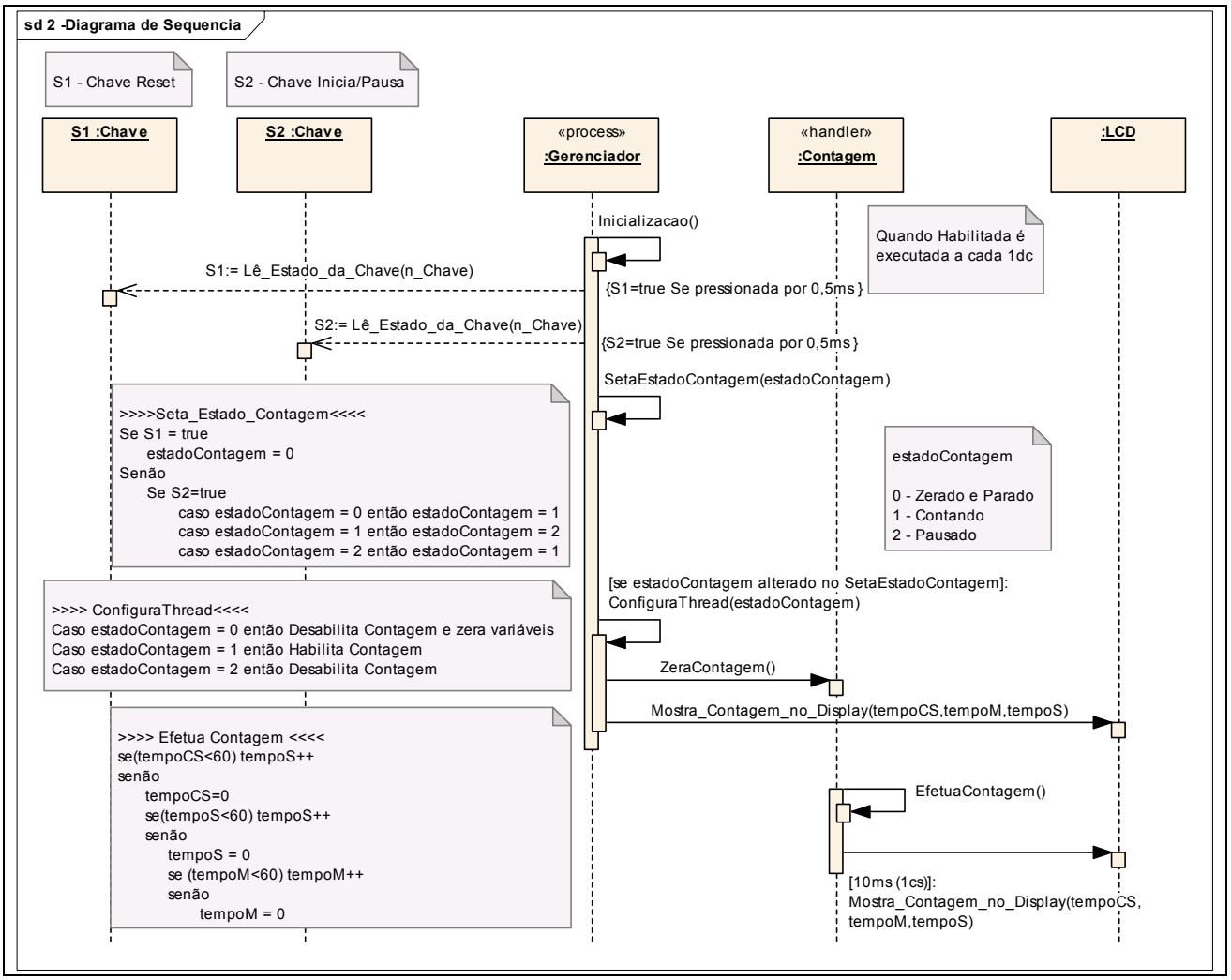

Figura 32. Diagrama de seqüência da aplicação Cronômetro

A Figura 33 apresenta o diagrama de estados da aplicação Cronômetro.

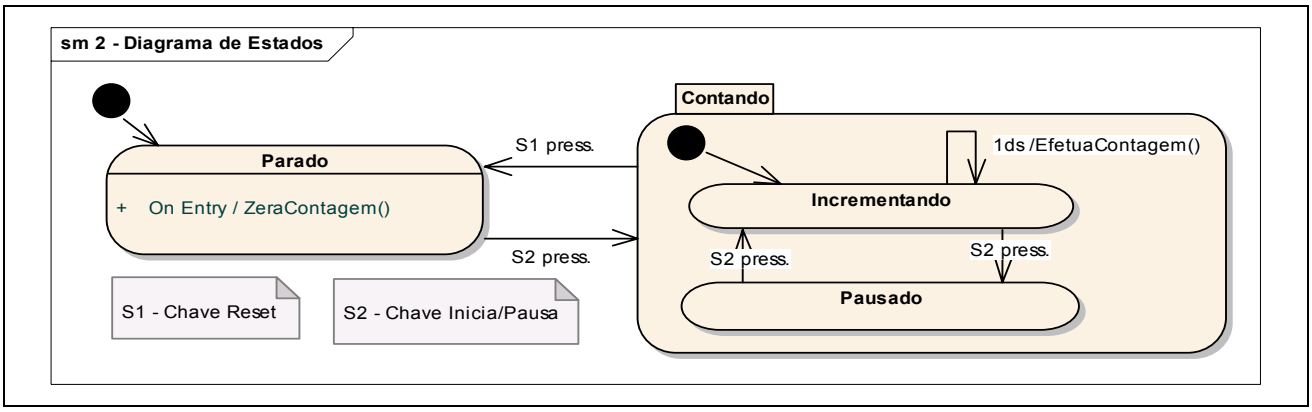

Figura 33. Diagrama de estado da aplicação Cronômetro

A Figura 34 mostra o fluxograma da Aplicação Cronômetro.

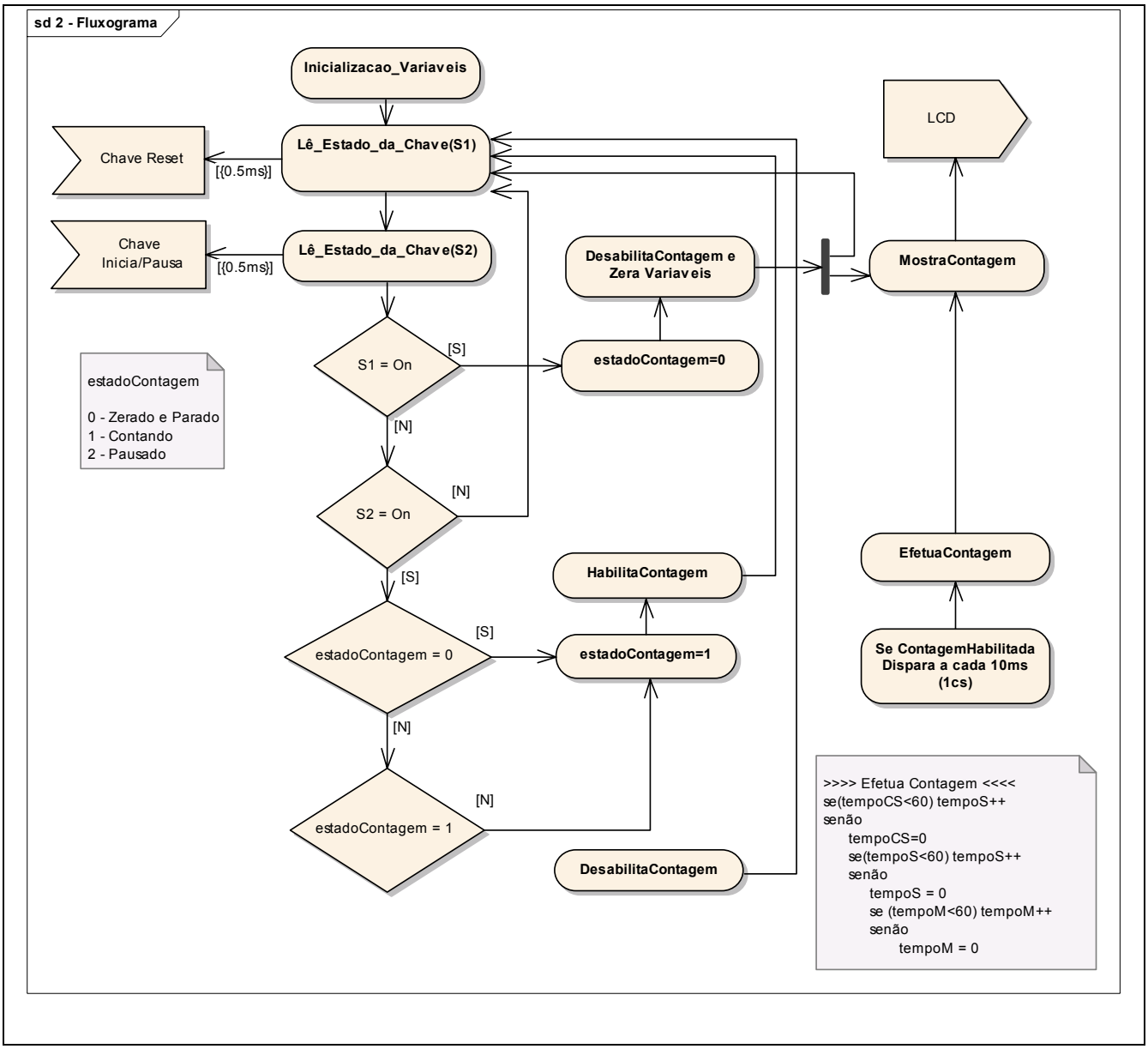

Figura 34. Fluxograma da aplicação Cronômetro

3.2.2. Projeto da Aplicação 2

3.2.2.1. Projeto segundo ROPES

Na fase de projeto, os componentes físicos e/ou lógicos do sistema são detalhados através do diagrama de componentes e construção. A Figura 35 apresenta os componentes da aplicação Cronômetro.

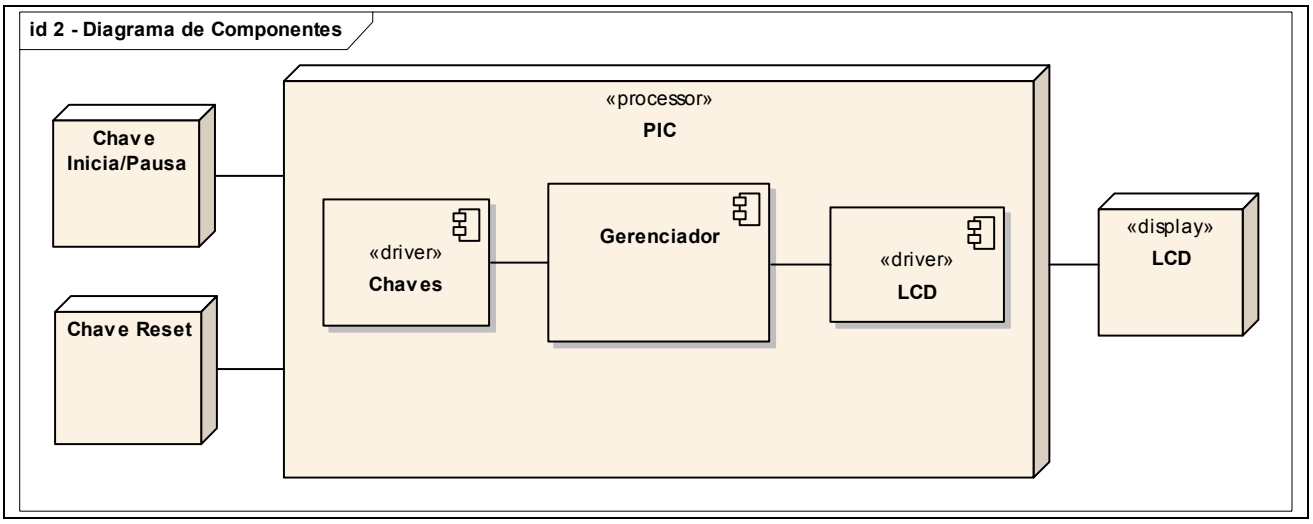

Figura 35. Diagrama de componentes e construção da aplicação Cronômetro

3.2.2.2. Projeto segundo WOLF

Como visto anteriormente esta fase é divida pela metodologia WOLF em duas etapas, a primeira elabora o diagrama em blocos do software e a segunda do hardware. A Figura 36 apresenta o Diagrama de Blocos do Software da aplicação Cronômetro.

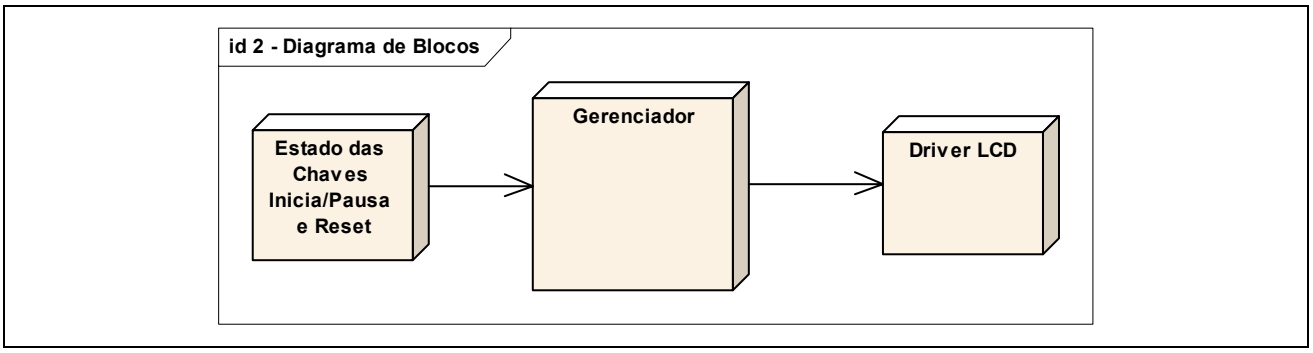

Figura 36. Diagrama de blocos do software da aplicação Cronômetro

A Figura 37 apresenta o Diagrama de Blocos do Hardware da aplicação Cronômetro.

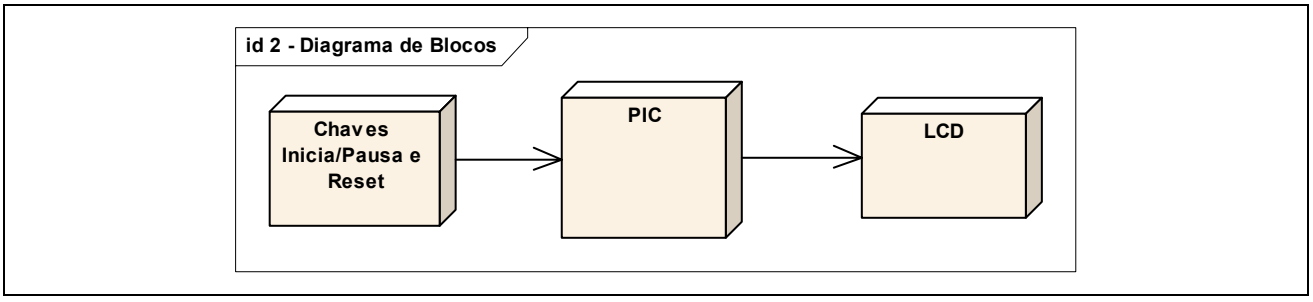

Figura 37. Diagrama de blocos do hardware da aplicação Cronômetro

3.2.3. Implementação e Teste da Aplicação 2

Para a implementação do esquema de hardware, teste e validação da aplicação Cronômetro, assim como na aplicação Testador de Display de 7 Segmentos, foi utilizada a ferramenta chamada PROTEUS/ISIS, a qual faz simulação de circuitos e componentes eletro-eletrônicos. A Figura 38 apresenta o esquema da ligação física do hardware dessa aplicação.

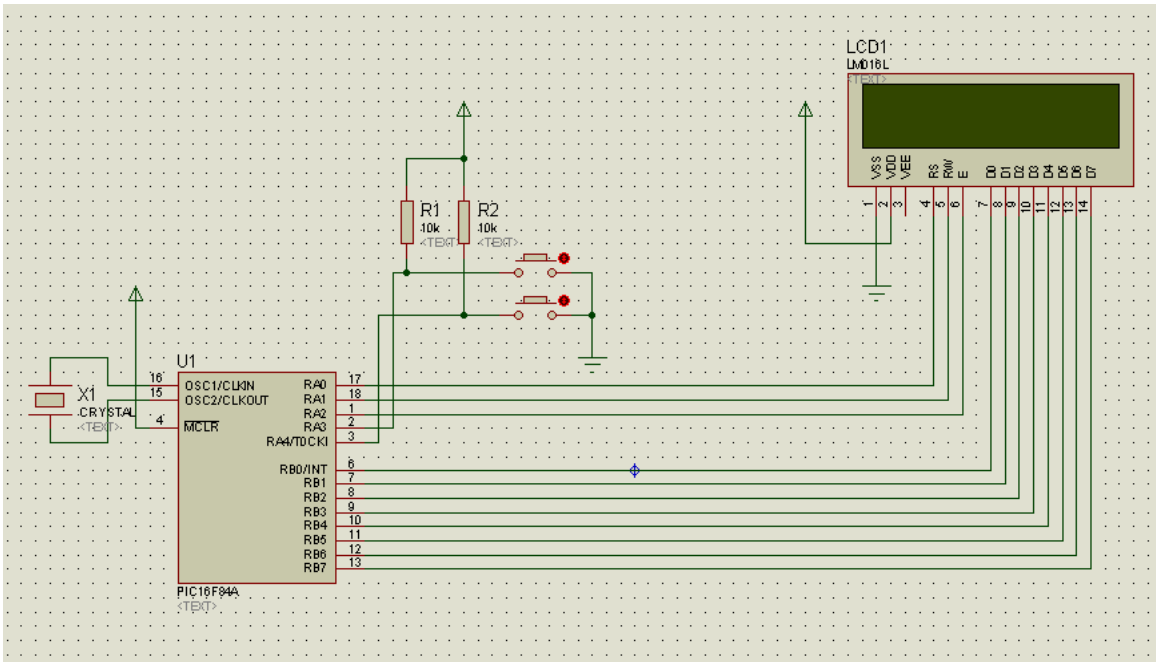

Figura 38. Esquema de ligação física do hardware da aplicação Cronômetro

O software da aplicação Cronômetro também foi desenvolvido com a utilização do compilador C PCWH da CCS. O software da aplicação foi testado e validado na ferramenta de simulação PROTEUS/ISIS, apresentado na Figura 38, e no Kit de prototipação disponível no LSED.

O trecho de código apresentado na Figura 39, representa o handler da aplicação que quando habilitada, efetua a contagem do tempo.

```
#int_timer0 // Timer responsável pela execução da Handler Contagem a cada 10ds
void contagem_timer0() {
  static int conta10ds = 0;
  static int conta1s  = 0;
  set_timer0(TMR0_INIT - get_timer0());
  conta10ds++;
  if (conta10ds == 100) {
```

```
    conta10ms = 0;
    if (tempoS<60) tempoS++;
    else {
      tempoS = 0;
      if (tempoM<60) tempoM++;
        else
          tempoM = 0;
    }
    sprintf(strtmp,"%02d:%02d",tempoM,tempoS);
    lcd_set_cursor_home();
    lcd_print(strtmp);
 }
}
```

Figura 39. Trecho de código da aplicação Cronômetro I

O trecho de código apresentado na Figura 40, é responsável pelo controle da thread Contagem mostrada acima.

```
void ConfiguraHandler (int estadoContagem) {

   switch (estadoContagem) {
      case 0: //desliga handler
              disable_interrupts (global | int_timer0);
              break;
      case 1: //inicia handler
              set_timer0        (TMR0_INIT);
              enable_interrupts (global | int_timer0);
              break;
      case 2: //paralisa handler
              timer = get_timer0 ();
              disable_interrupts (global | int_timer0);
              break;
      case 3: //liga handler
              set_timer0        (timer);
              enable_interrupts (global | int_timer0);
              break;
  }
}
```

Figura 40. Trecho de código da aplicação Cronômetro II

3.3. APLICAÇÃO 3: MONITORAMENTO DE DADOS REMOTOS

A terceira aplicação, esquematizada na Figura 41, possui um nível de dificuldade maior de implementação para uma melhor análise das técnicas e metodologias estudadas. Considerando o tempo disponível para a realização deste TCC, optou-se por utilizar uma aplicação já desenvolvida no LSED, de forma a otimizar o tempo de implementação da mesma. A aplicação selecionada como referência foi a Aplicação Monitoramento de Dados Remotos desenvolvida por Carlos Cimolin (2004).

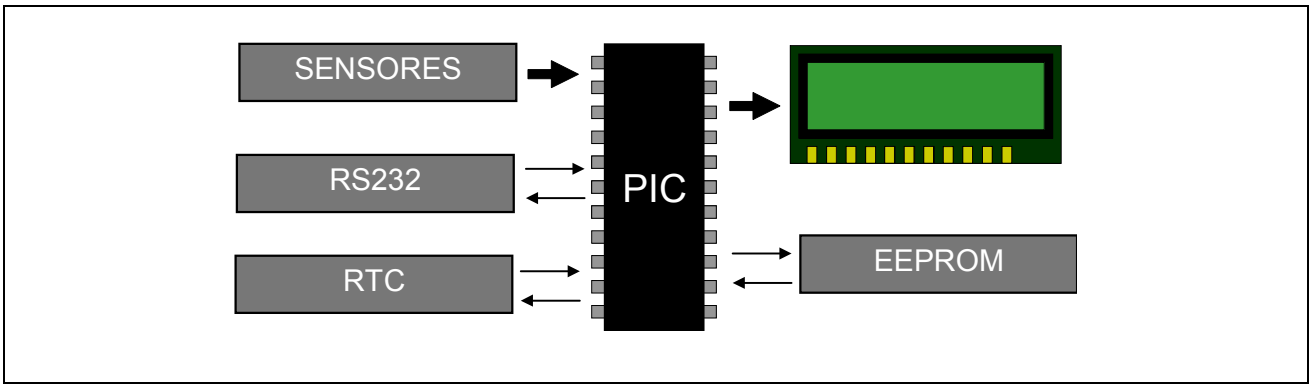

Figura 41. Aplicação Monitoramento de Dados Remotos

Carlos Cimolin (2004) resume a aplicação como sendo um sistema embarcado para medição e transmissão de dados remotamente. Esses dados são coletados por uma plataforma remota, analisados e empacotados para serem enviados através de telefonia para uma central, onde serão extraídos e disponibilizados para visualização e análise.

O autor modelou essa aplicação utilizando uma combinação de conceitos da metodologia WOLF com a técnica DFD. A aplicação foi implementada sobre uma plataforma baseada no microcontrolador PIC, porém a validação foi limitada à simulação de um modelo dessa plataforma no simulador PROTEUS/ISIS.

3.3.1. Requisitos da Aplicação 3

3.3.1.1. Requisitos segundo ROPES

Análise de requisitos

Para facilitar a visualização dos requisitos da aplicação, o diagrama de caso uso foi elaborado com três níveis de abstração.

O caso de uso geral da aplicação Monitoramento de dados Remotos apresentado na Figura 42, efetua o monitoramento dos dados remotos, coletando amostras de variáveis ambientais através de sensores, durante um intervalo de tempo predefinido, calculando a média das amostras de cada variável e registrando as médias, a data e o horário do monitoramento na EEPROM. Se necessário disponibiliza informações da execução no LCD. Comunica-se com a Estação Base para obter a configuração do sistema (data, hora, parâmetros do monitoramento) e para disponibilizar os dados registrados.

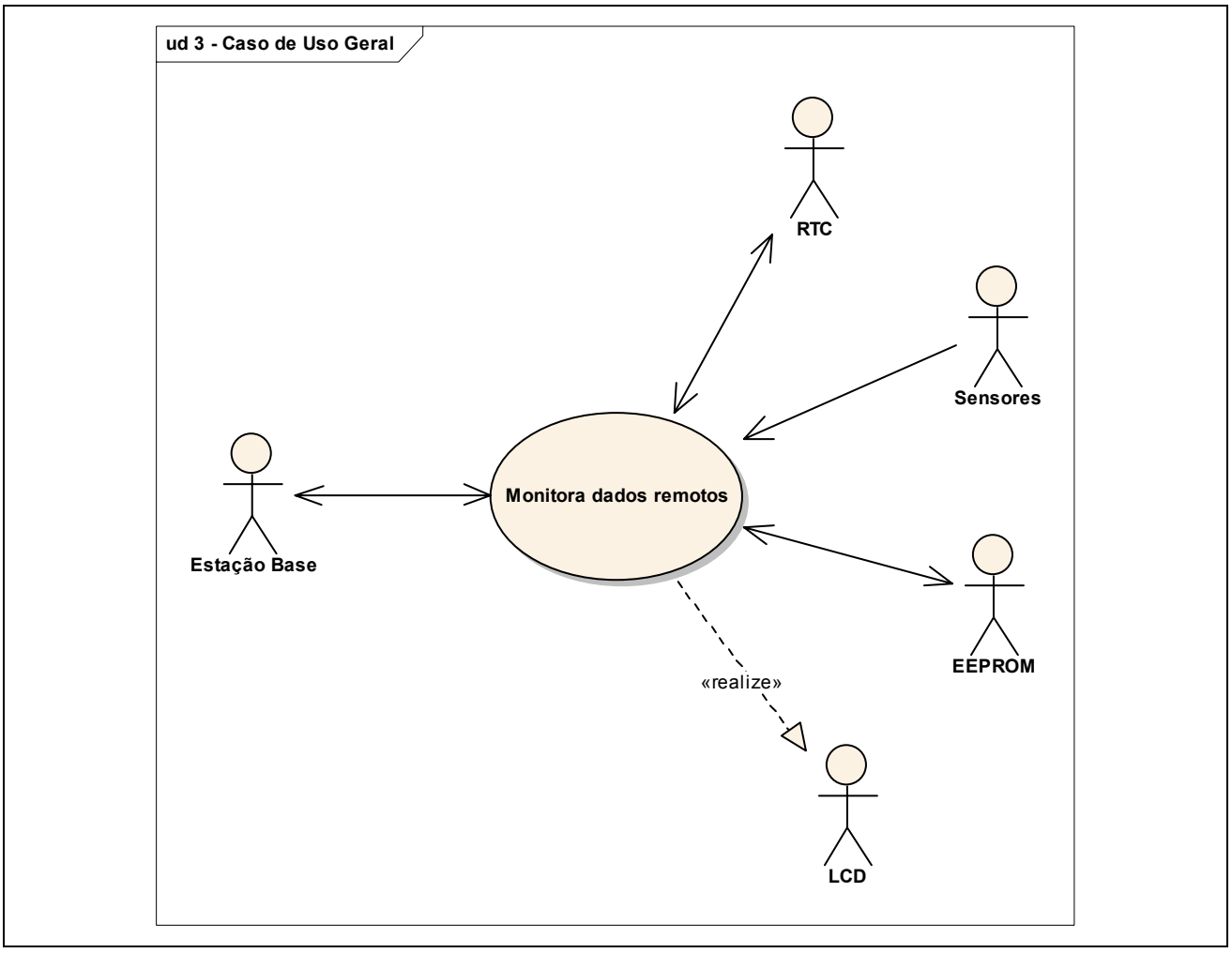

Figura 42. Diagrama de caso de uso geral da aplicação Monitoramento de Dados Remotos

No diagrama de caso de uso detalhado, apresentado na Figura 43, a estação base envia a informação da data e do horário que é encaminhada ao RTC, efetuando a sua configuração. Ela também envia os parâmetros do monitoramento que devem ser registrados na EEPROM interna.

Os parâmetros de monitoramento incluem:

- Intervalo de tempo entre monitoramentos sucessivos (em minutos)
- Período de monitoramento (em minutos)
- Taxa de amostragem (número de amostras por minuto)
- Variáveis por amostra (número de sensores lidos)
- Sensores a serem lidos

O caso de uso “Realiza o monitoramento”, efetua o monitoramento dos dados remotos, lendo os sensores durante um período de monitoramento definido e com uma taxa de amostragem pré-estabelecida. Calcula a média das amostras de cada sensor e as registra, na EEPROM, incluindo a data e o horário do monitoramento, obtidos do RTC. Se necessário, disponibiliza informações da execução no LCD.

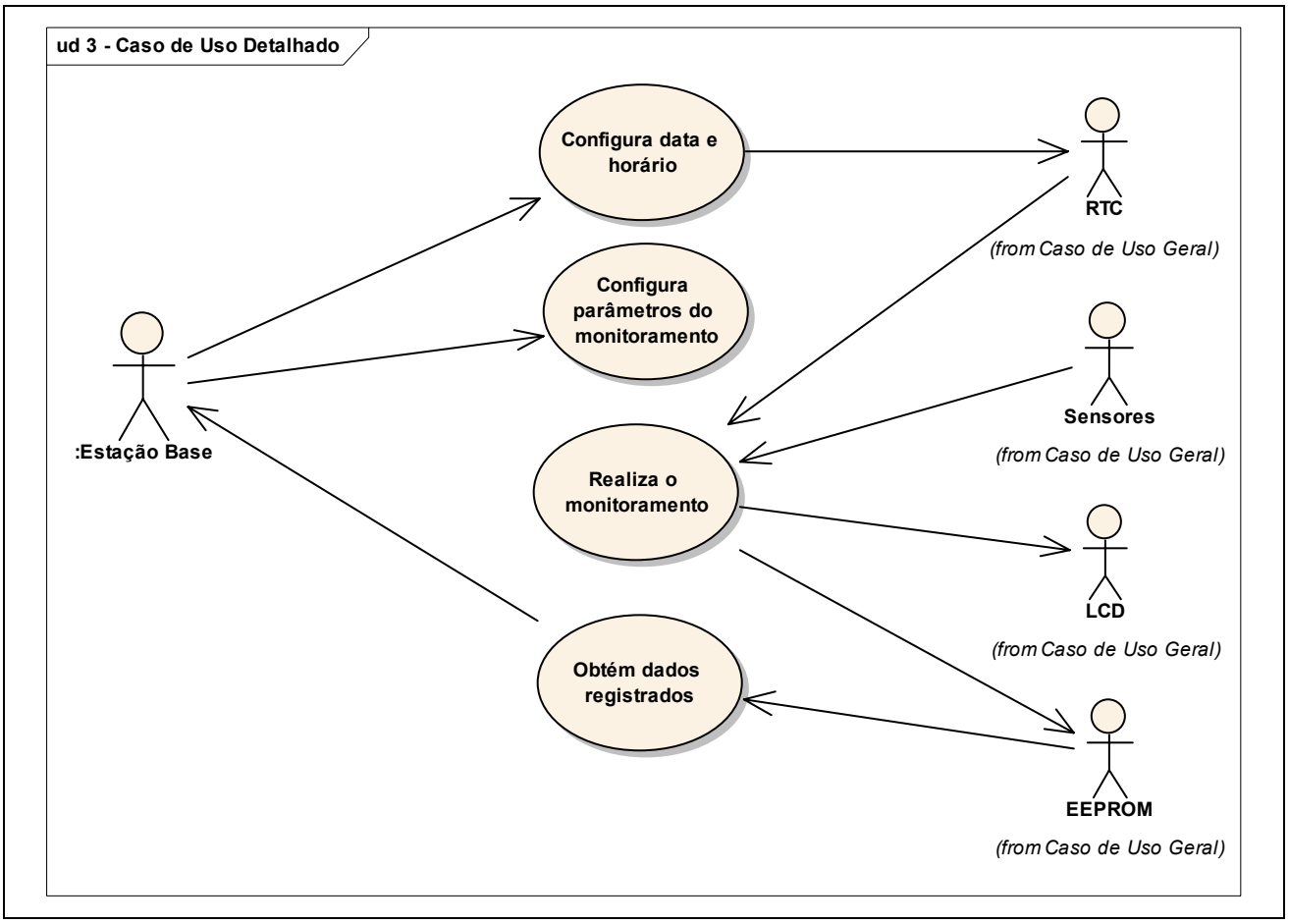

Figura 43. Diagrama de caso de uso detalhado da aplicação Monitoramento de Dados Remotos

O intervalo de tempo entre monitoramentos é utilizado para configurar um timer interno que desperta o sistema no momento de execução do próximo monitoramento. Ao executar o monitoramento, é utilizado um contador interno para definir os instantes de amostragem com base na taxa configurada. Nos intervalos de tempo entre monitoramentos e amostragem o sistema deve ir para modo de economia de energia.

A estação_base também pode solicitar a transferência (*upload*) dos dados registrados na EEPROM externa. Após a transferência, a área de memória utilizada por esses dados é liberada.

Na Figura 44, o caso de uso “Monitoramento” é refinado e explorado. Ele lê os sensores durante um período monitoramento, que é definido por um intervalo de tempo, que configura um timer interno despertando o sistema no momento de execução de cada leitura. Ao executar o monitoramento é utilizado um contador interno para definir os instantes de amostragem com base na taxa configurada.

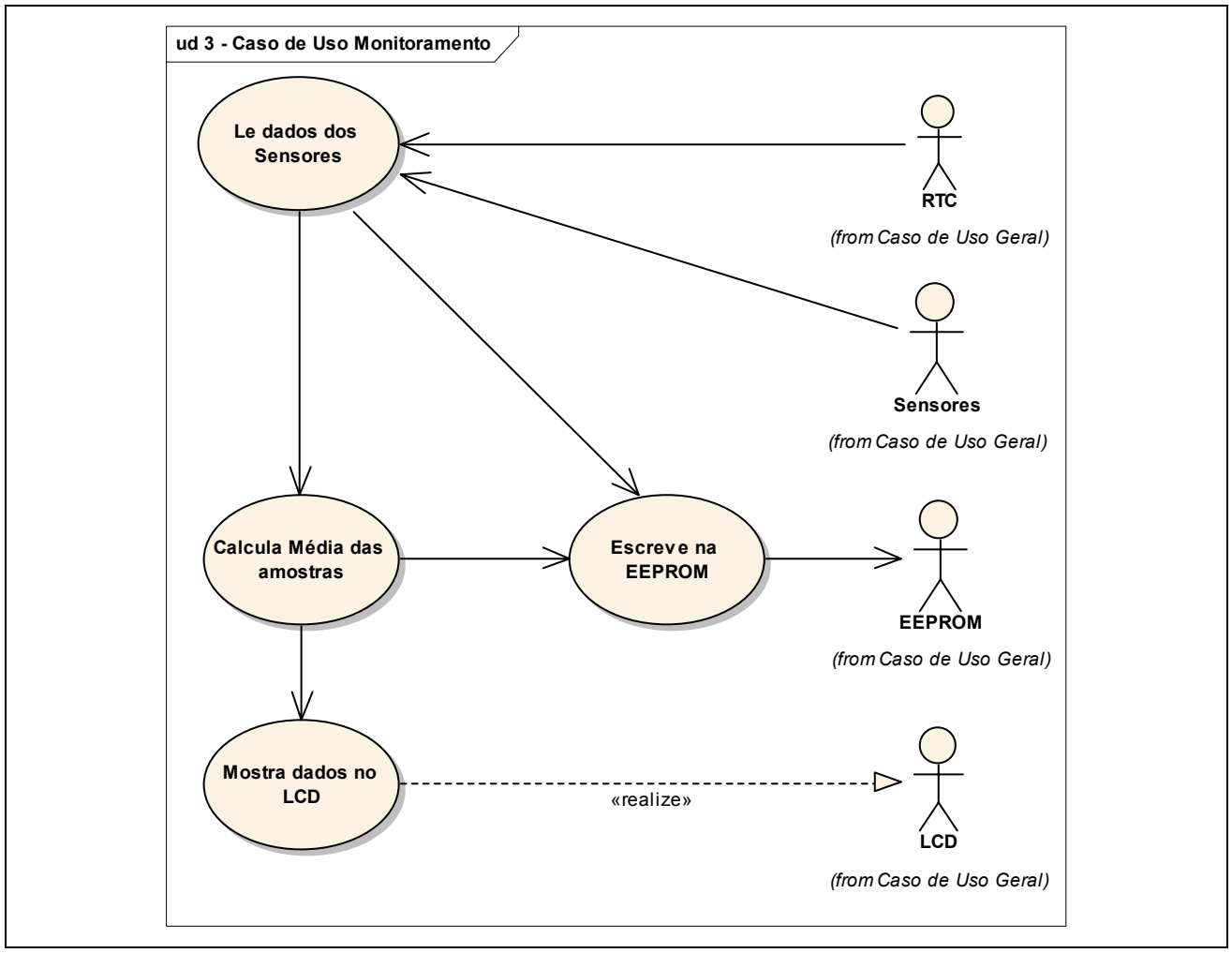

Figura 44. Diagrama de caso de uso da aplicação Monitoramento de Dados Remotos

Durante cada amostragem, o sistema deve coletar uma amostra de cada sensor (2 bytes), registrando-as em um buffer na memória EEPROM externa (ex. 1KByte). Essas amostras são usadas no cálculo das médias que após, então, são registradas na EEPROM externa. O monitoramento de dados deve ser bloqueado quando a EEPROM externa é preenchida.

O sistema calcula a média das amostras e encaminha os dados para a escrita na EEPROM. Se necessário, solicita a apresentação desses dados no LCD. Registra cada amostragem na

EEPROM externa. Registra a média, a data e o horário das amostras na EEPROM externa. O registro de dados deve ser bloqueado quando a EEPROM externa é preenchida.

Análise de objetos

O diagrama de classes da aplicação é apresentado na Figura 45.

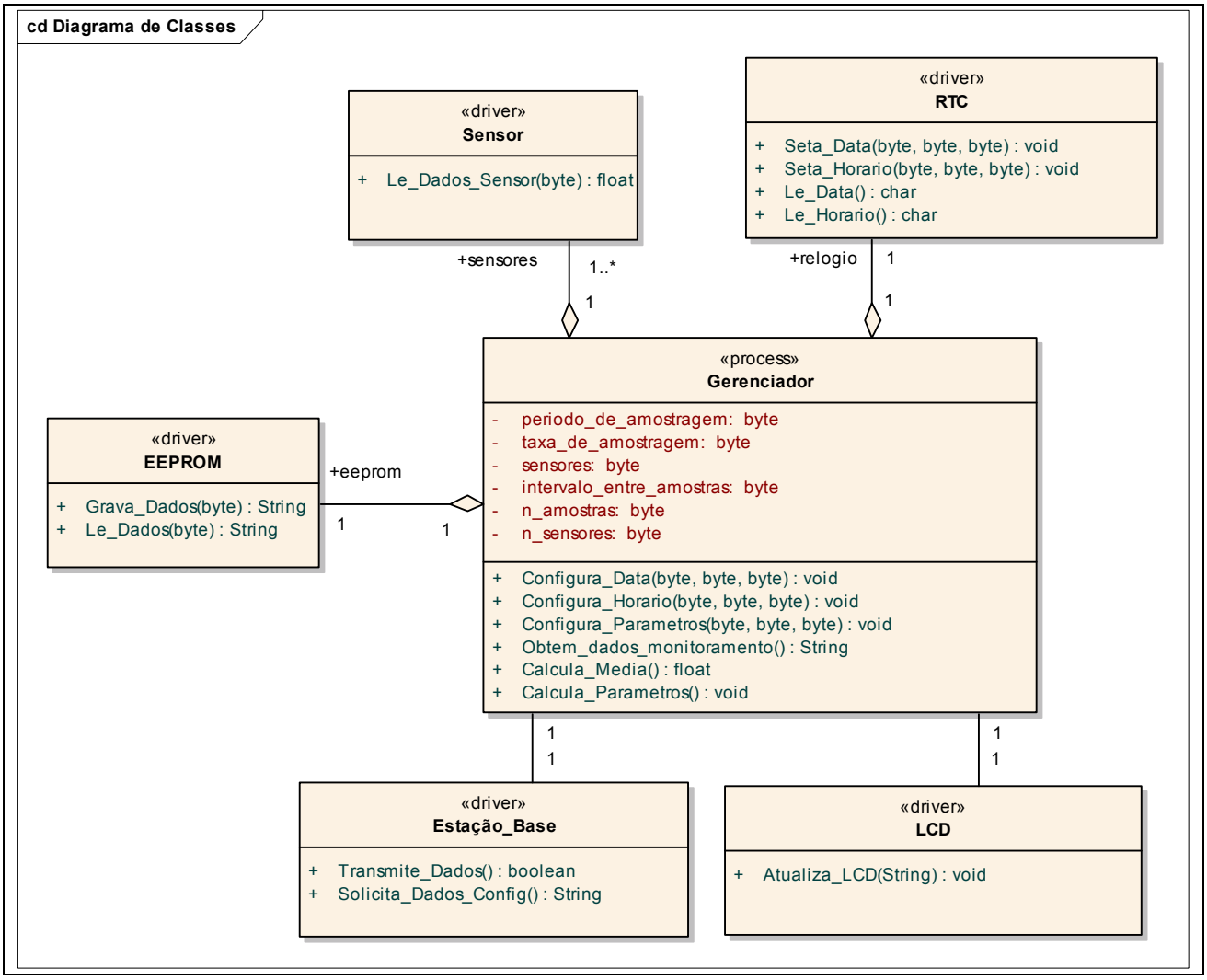

Figura 45. Diagrama de classes da aplicação Monitoramento de Dados Remotos

O detalhamento do comportamento dos objetos do sistema é descrito a seguir pelo diagrama de seqüência. Para facilitar sua visualização, o diagrama está dividido em duas partes.

A Figura 46, apresenta o diagrama de seqüência da aplicação Monitoramento de Dados Remotos, onde é descrito a comunicação da estação base e a aplicação.

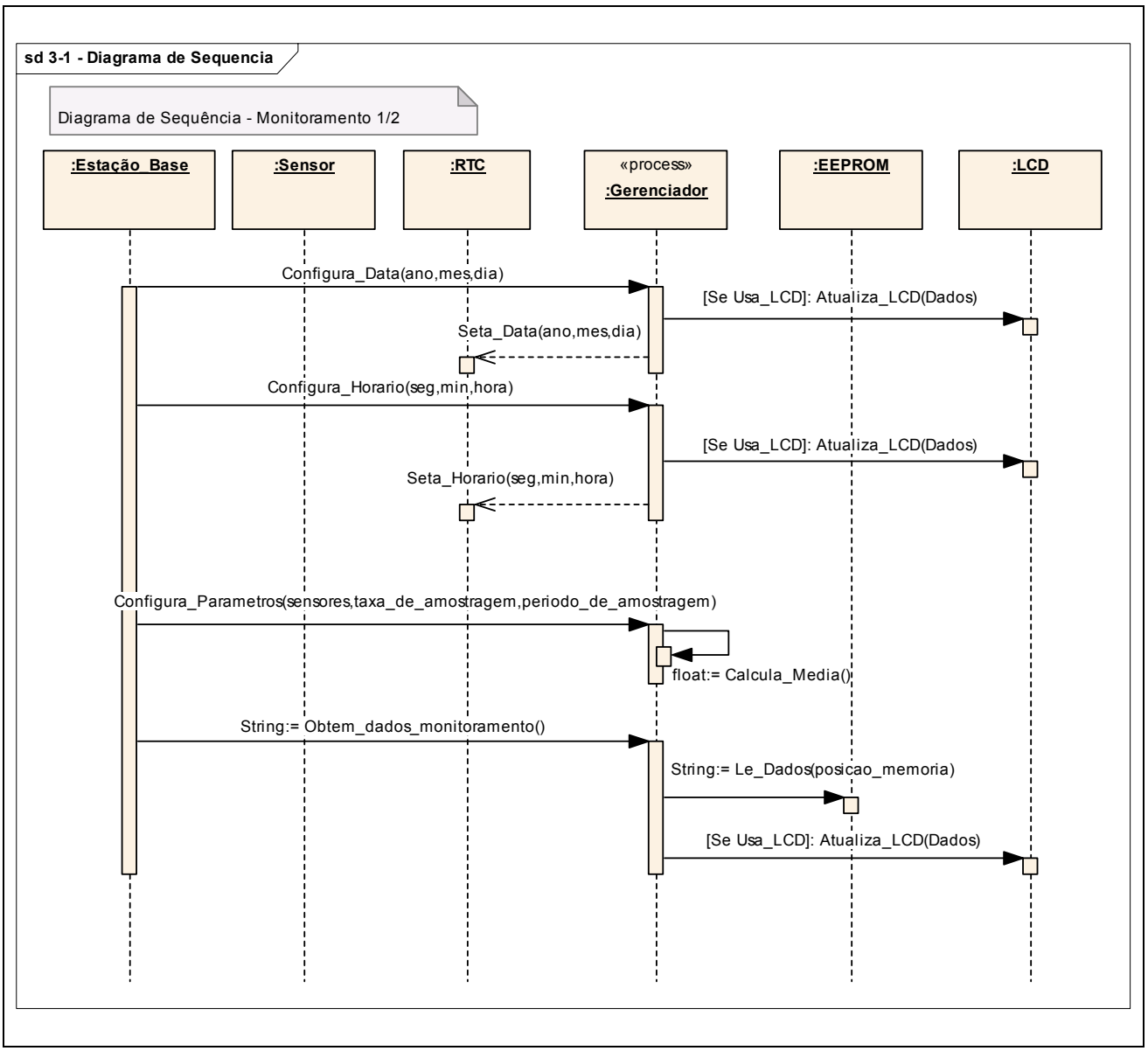

Figura 46. Primeira parte do diagrama de seqüência da aplicação Monitoramento de Dados Remotos

A Figura 47, apresenta o diagrama de seqüência da aplicação Monitoramento de Dados Remotos, onde é descrito o monitoramento dos dados remotos.

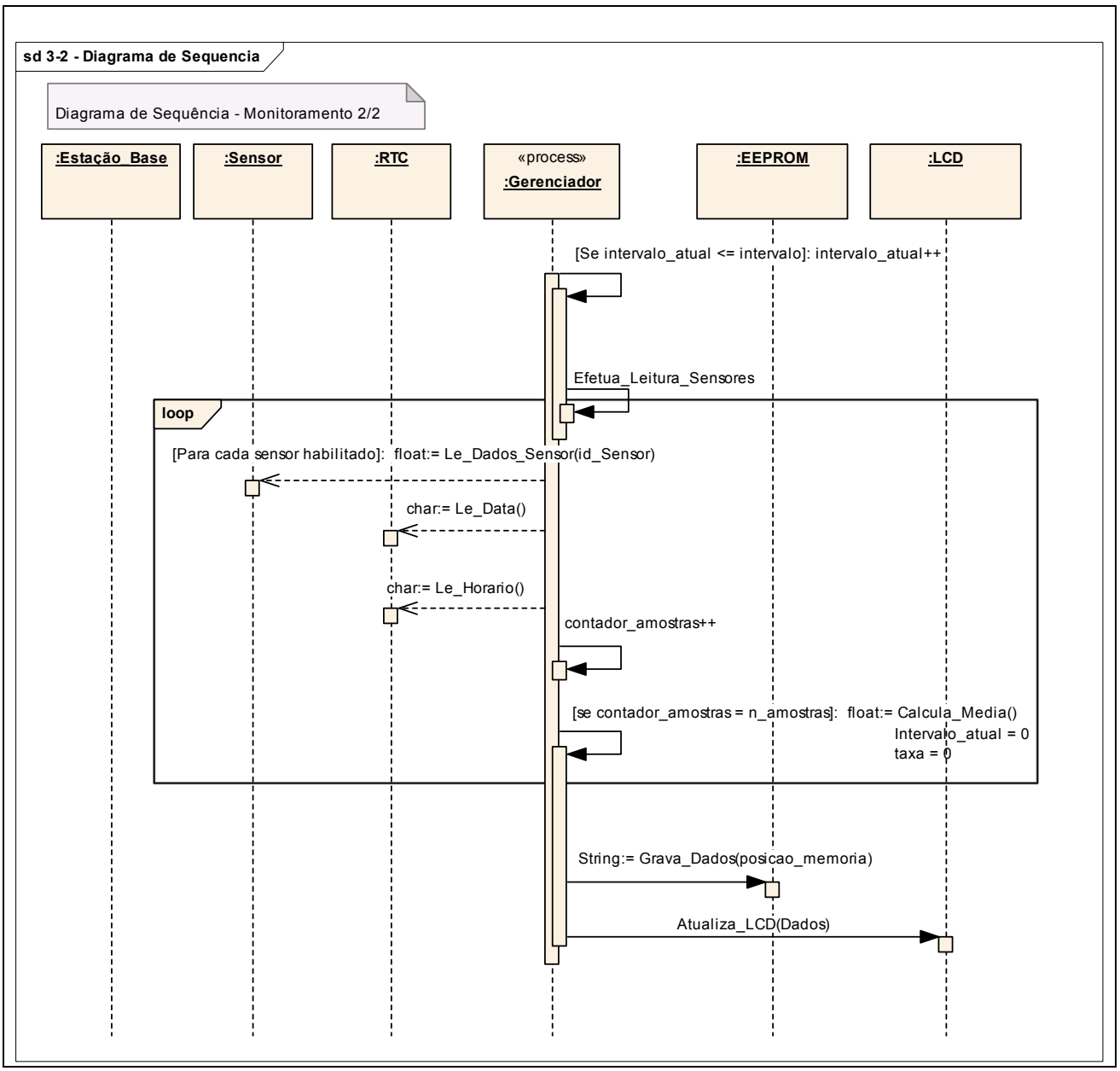

Figura 47. Segunda parte do diagrama de seqüência da aplicação Monitoramento de Dados Remotos

3.3.1.2. Requisitos segundo WOLF

Requisitos

Os requisitos básicos desta aplicação estão descritos abaixo:

- Finalidade: aquisição de dados de sensores analógicos, processamento e transmissão através de um meio de comunicação para uma estação central baseada em um computador pessoal;
- Entrada: sinais adquiridos vindos de sensores analógicos, comandos recebidos de uma estação central (PC) e chave de *reset* da aplicação;
- Saídas: dados pós-processados transmitidos para uma estação central, indicadores luminosos de funcionamento do protótipo (*on/off*) e um display de cristal líquido (LCD); e
- Função: adquirir dados de vários sensores, com várias taxas de amostragem, processar esses dados, armazená-los e transmití-los para uma estação central (PC).

Especificação

Nesta fase é feito um detalhamento maior dos requisitos da aplicação, como apresentados a seguir:

- Finalidade: adquirir dados de sensores instalados remotamente, fazendo um pré-processamento dos dados, o armazenamento temporário desses dados e a transmissão dos mesmos para uma estação central;
- Entrada: sinais analógicos vindos dos sensores e sinais digitais lidos da EEPROM externa e sinais digitais vindos dos parâmetros de configuração recebidos por meio do sistema de comunicação
- Saída: sinais digitais, dados escritos na EEPROM externa e dados enviados por meio do sistema de comunicação;
- Funções: Adquirir dados de sensores analógicos com diferentes taxas de amostragens; Gravar e ler dados em uma memória não volátil; Calcular a média dos dados a serem transmitidos; Permitir alterar a configuração de alguns parâmetros, através de comandos recebidos da estação central; Manter um relógio de tempo-real, sincronizado com o relógio da estação central; Utilizar protocolo de comunicação do tipo I^2C com dispositivos externos; Possuir capacidade de armazenar até 32Kbits de dados em memória não-volátil; Disponibilizar *reset,* através da chave de *reset*.

O diagrama de classes da aplicação é apresentado na Figura 45.

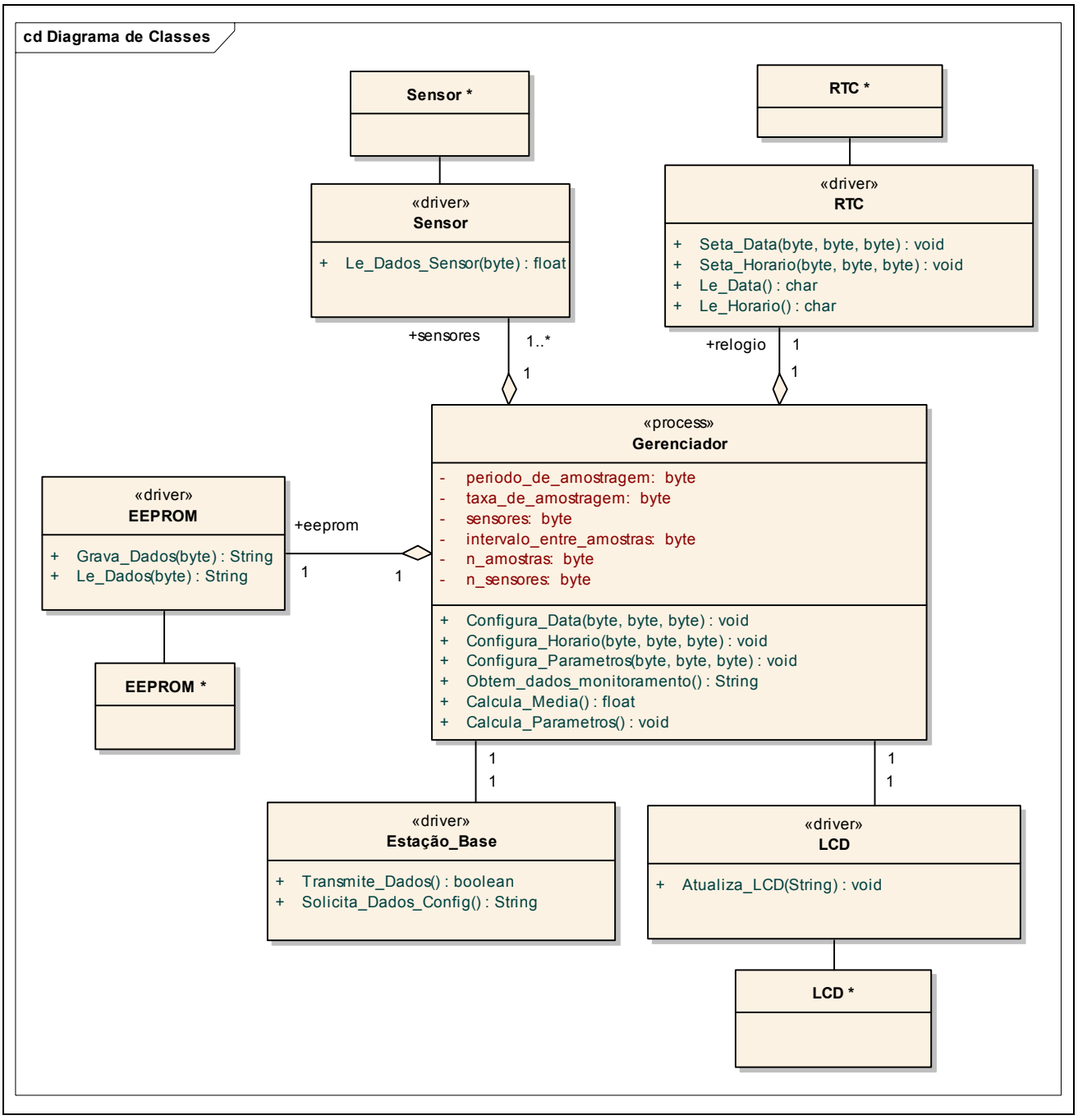

Figura 48. Diagrama de classes da aplicação Monitoramento de Dados Remotos

O detalhamento do comportamento dos objetos do sistema é descrito a seguir pelo diagrama de seqüência. Para facilitar sua visualização, o diagrama está dividido em duas partes.

A Figura 46, apresenta o diagrama de seqüência da aplicação Monitoramento de Dados Remotos, onde é descrito a comunicação da estação base e a aplicação.

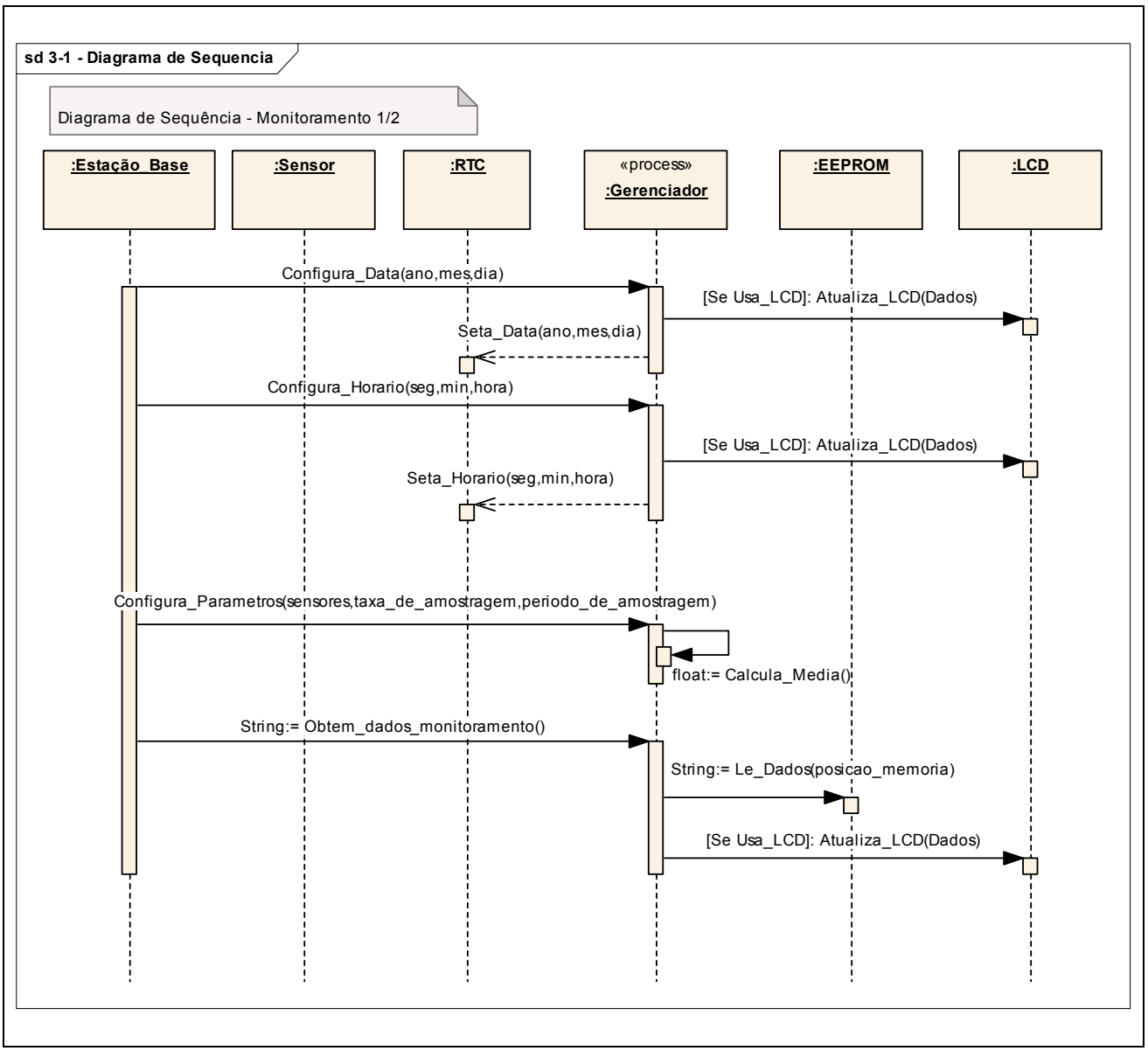

Figura 49. Primeira parte do diagrama de seqüência da aplicação Monitoramento de Dados Remotos

A Figura 47, apresenta o diagrama de seqüência da aplicação Monitoramento de Dados Remotos, onde é descrito o monitoramento dos dados remotos.

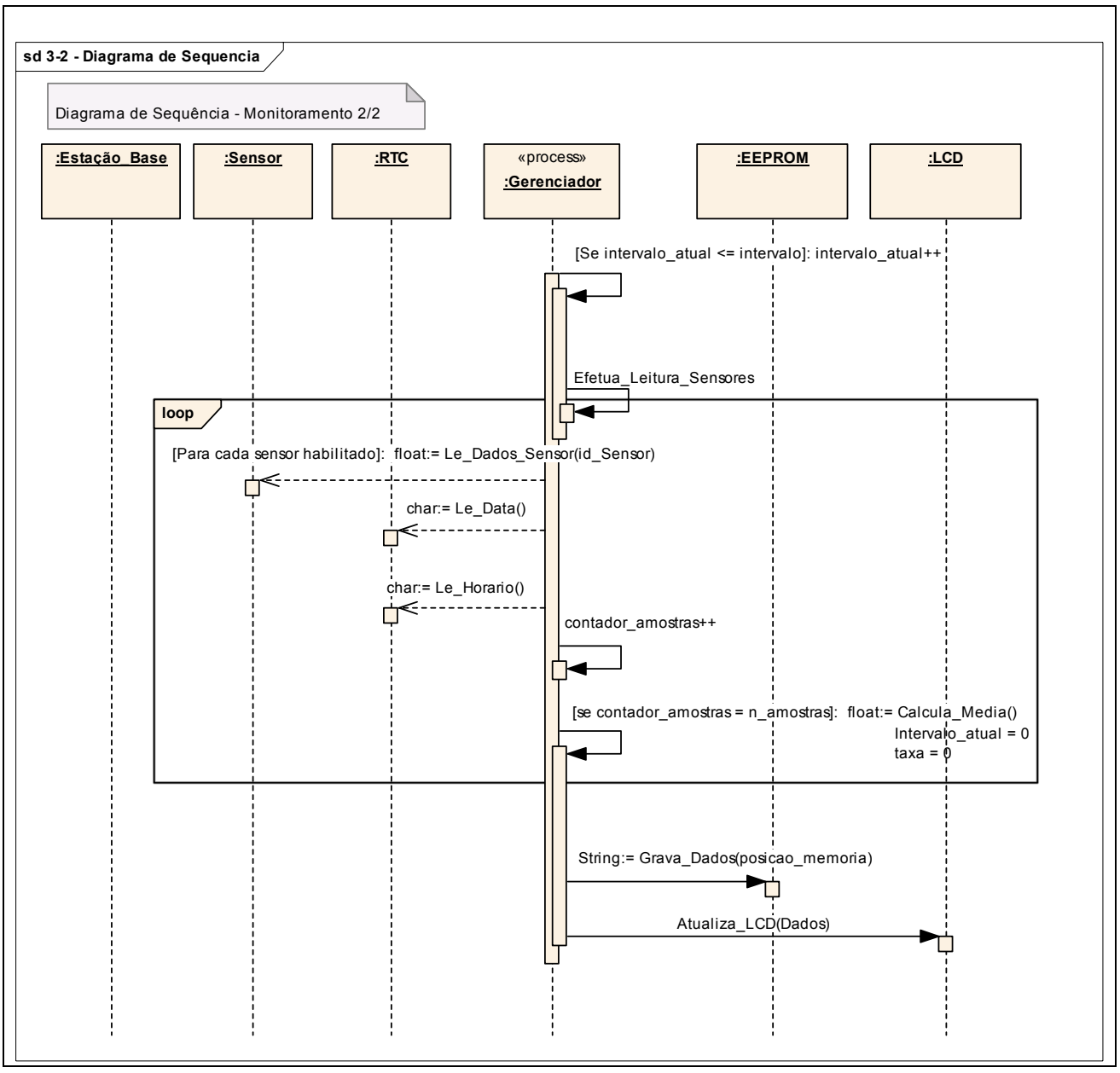

Figura 50. Segunda parte do diagrama de seqüência da aplicação Monitoramento de Dados Remotos

A Figura 51 apresenta o diagrama de estados da aplicação Monitoramento de dados Remotos.

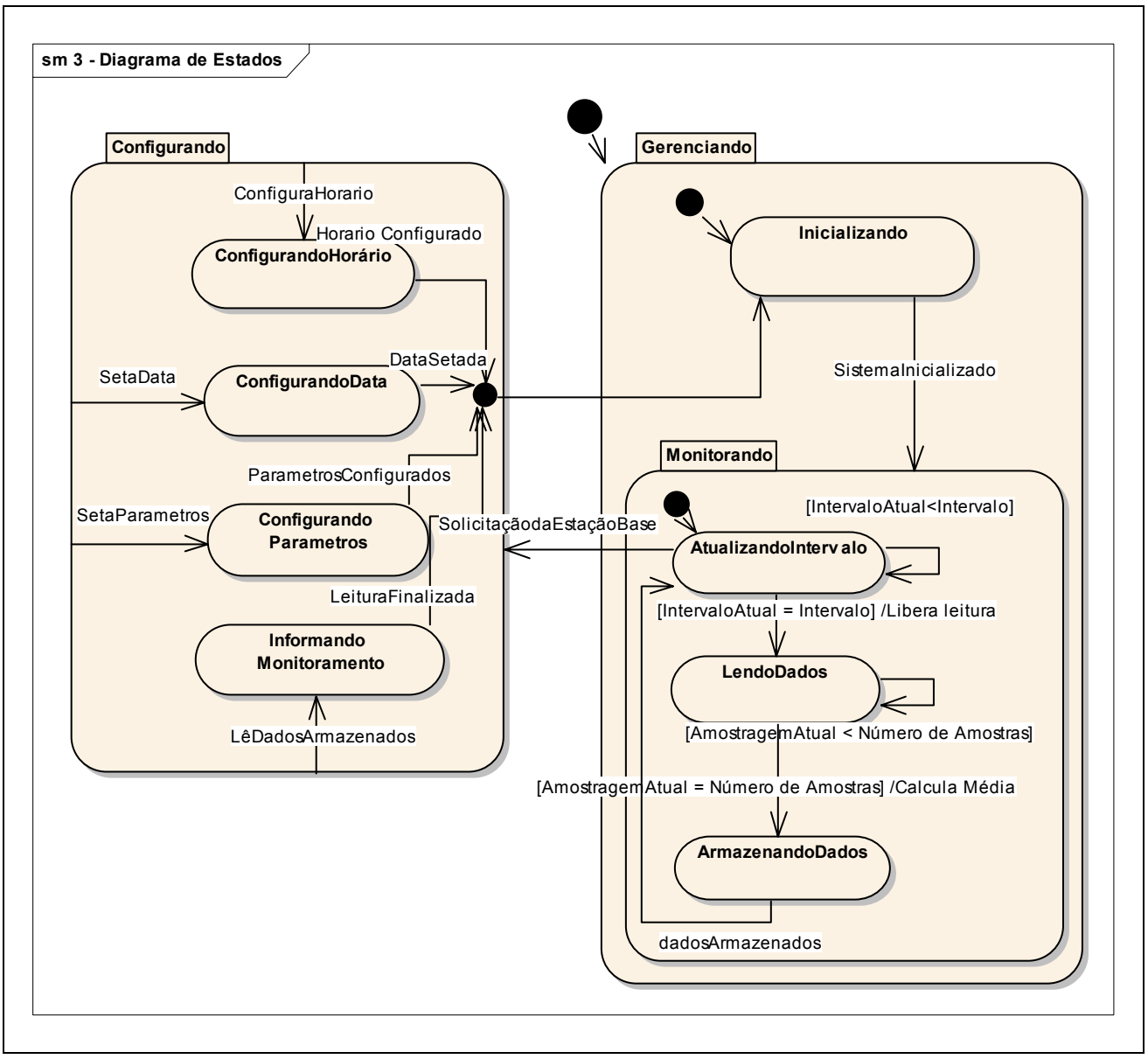

Figura 51. Diagrama de classes da aplicação Monitoramento de Dados Remotos

O fluxograma da aplicação Monitoramento de Dados Remotos, é apresentado a seguir. Para facilitar sua visualização, o fluxograma está dividido em duas partes.

A Figura 52, apresenta o fluxograma da aplicação Monitoramento de Dados Remotos, onde é descrito a comunicação entre a estação base e a aplicação.

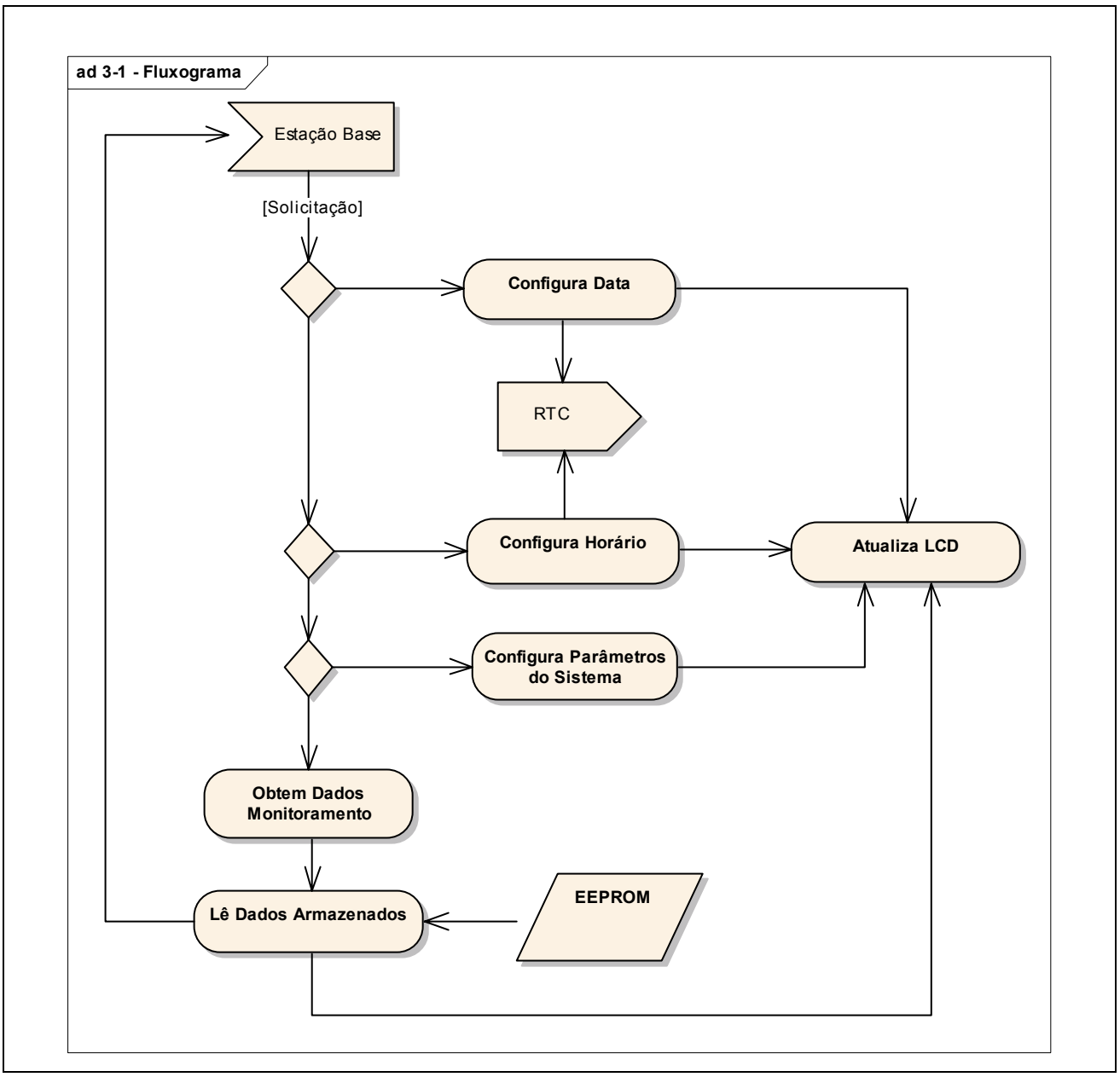

Figura 52. Primeira parte do fluxograma da aplicação Monitoramento de Dados Remotos

A Figura 53 apresenta o fluxograma da aplicação Monitoramento de Dados Remotos, onde é descrito o monitoramento dos Dados Remotos.

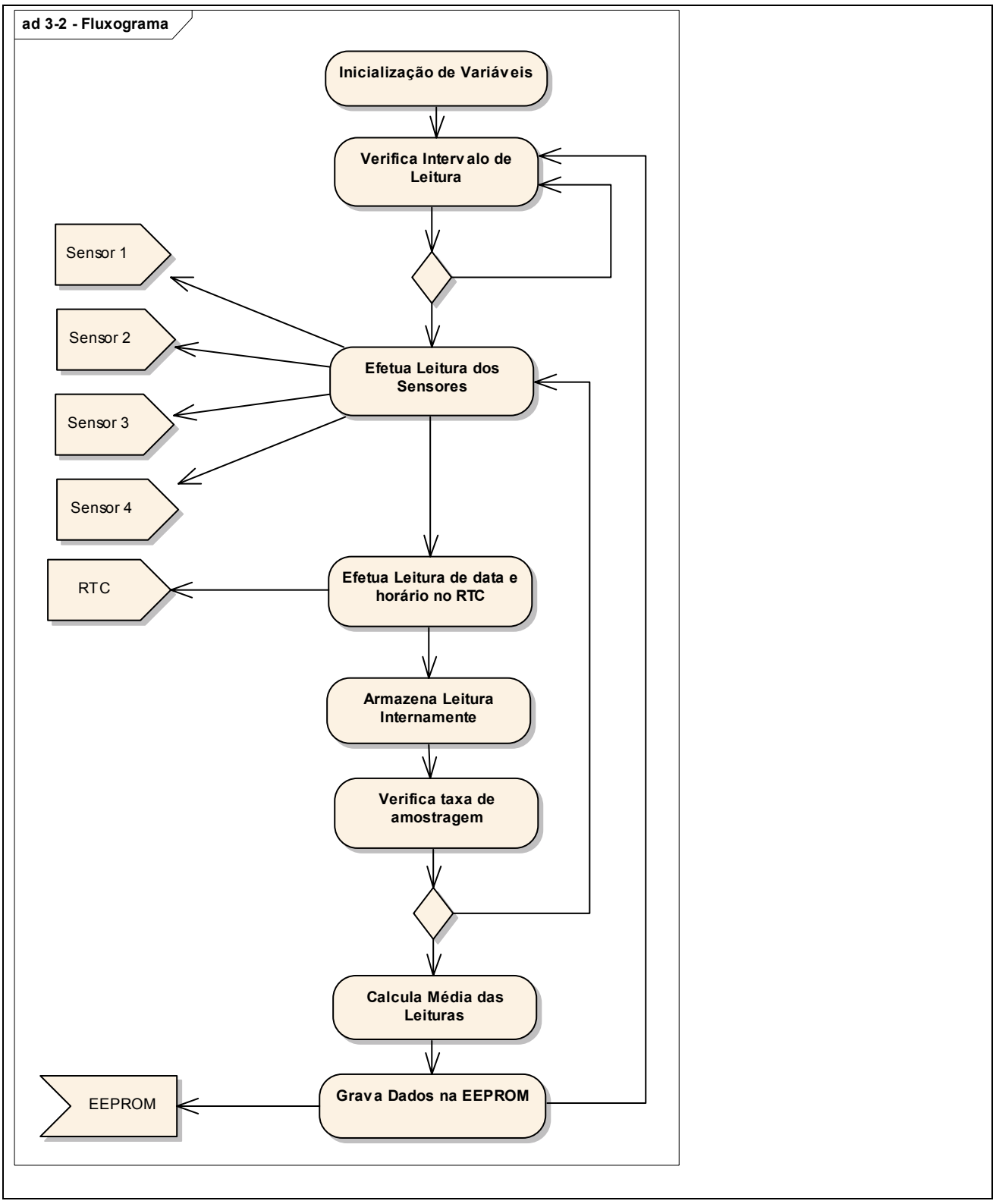

Figura 53. Segunda parte do fluxograma da aplicação Monitoramento de Dados Remotos

3.3.2. Projeto da Aplicação 3

3.3.2.1. Projeto segundo ROPES

A Figura 54 apresenta o diagrama de componentes e construção da aplicação Monitoramento de Dados Remotos.

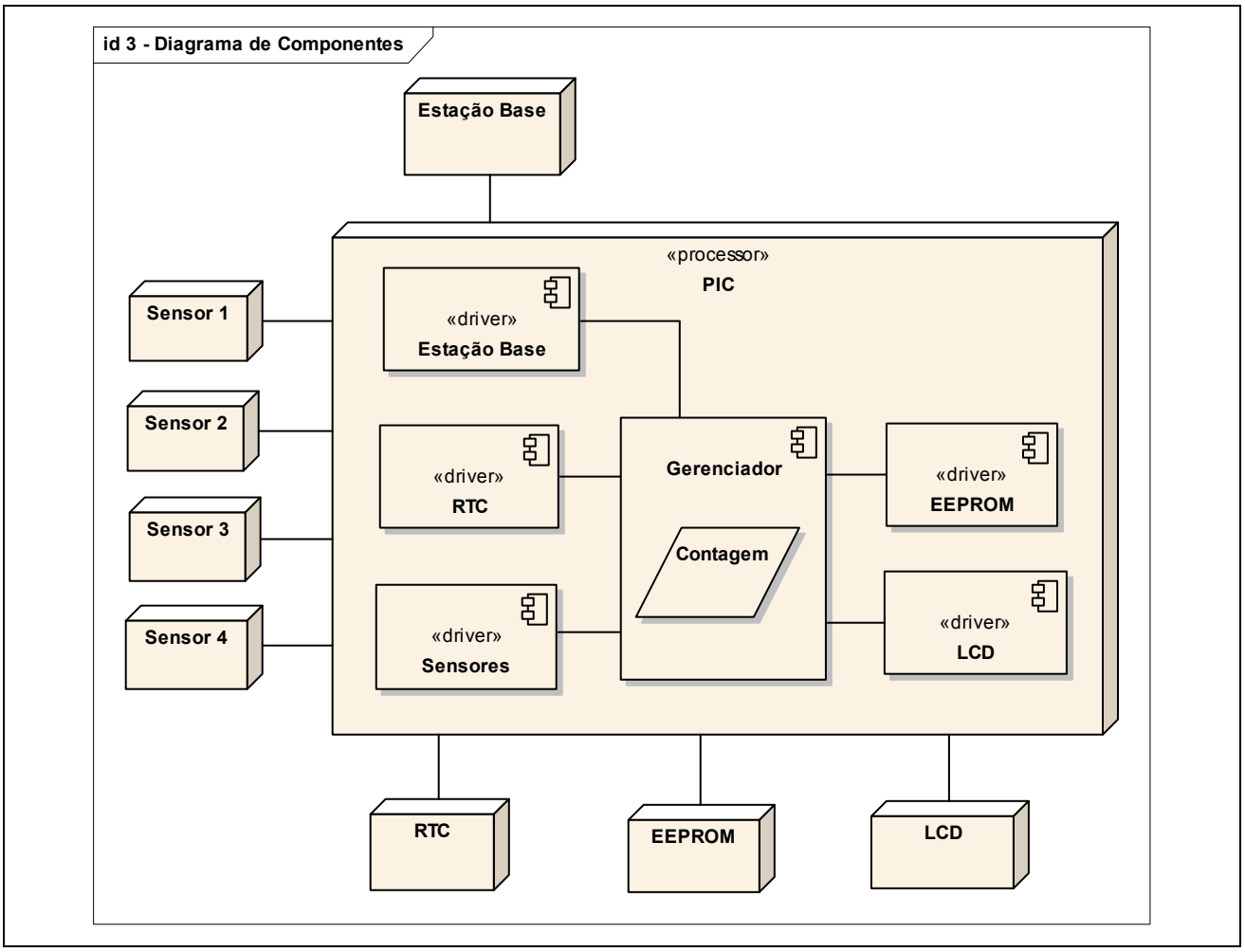

Figura 54. Diagrama de componentes e construção da aplicação Monitoramento de Dados Remotos

3.3.2.2. Projeto segundo WOLF

O projeto arquitetural do sistema é dividido em duas partes, o software e o hardware. Na Figura 55 é visualizado o diagrama em blocos da arquitetura de software, a qual é composta pelos componentes de software como, por exemplo, o gerenciador, o driver dos sensores, o driver do LCD, o driver da EEPROM, o driver do RTC e o driver da Estação Base .

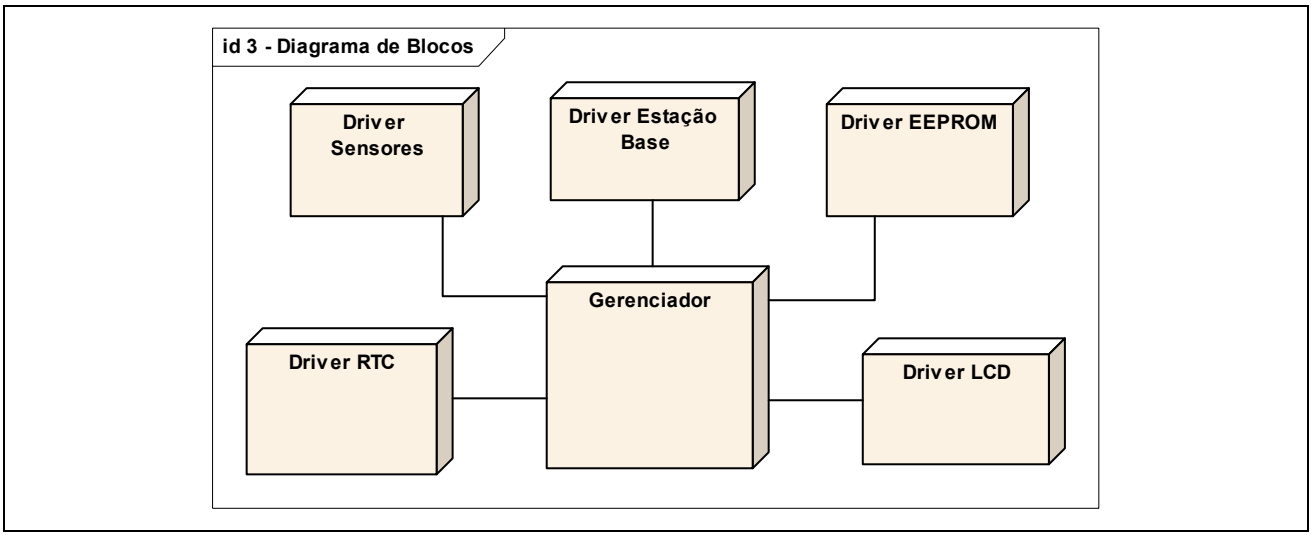

Figura 55. Diagrama de blocos do software da aplicação Monitoramento de Dados Remotos

Na Figura 56 é apresentado o diagrama de blocos do hardware da aplicação Monitoramento de Dados Remotos, onde tem-se, o microcontrolador e os periféricos externos, tais como sensores, display de cristal líquido (LCD), EEPROM serial, relógio de tempo real (RTC) serial e um dispositivo para comunicação com a Estação Base (RS-232).

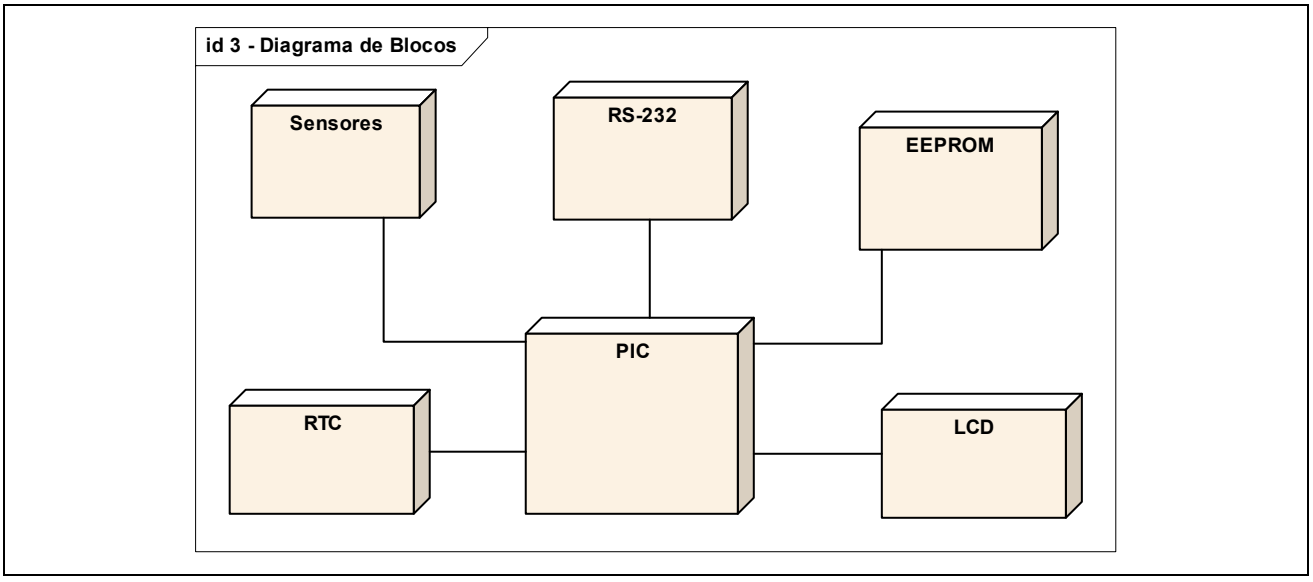

Figura 56. Diagrama de blocos do hardware da aplicação Monitoramento de Dados Remotos

3.3.3. Implementação e Teste da Aplicação 3

A aplicação Monitoramento de Dados remotos, desenvolvida por Carlos Cimolin (2004), foi realizada utilizando um microcontrolador interligado a três potenciômetros analógicos e um gerador de sinal senoidal, os quais representam os sensores; um relógio de tempo, uma memória EEPROM externa, um componente que faz a comunicação serial RS-232; e um display de cristal líquido.

Abaixo, na Figura 57, é visualizado o esquema de ligação física do hardware da aplicação.

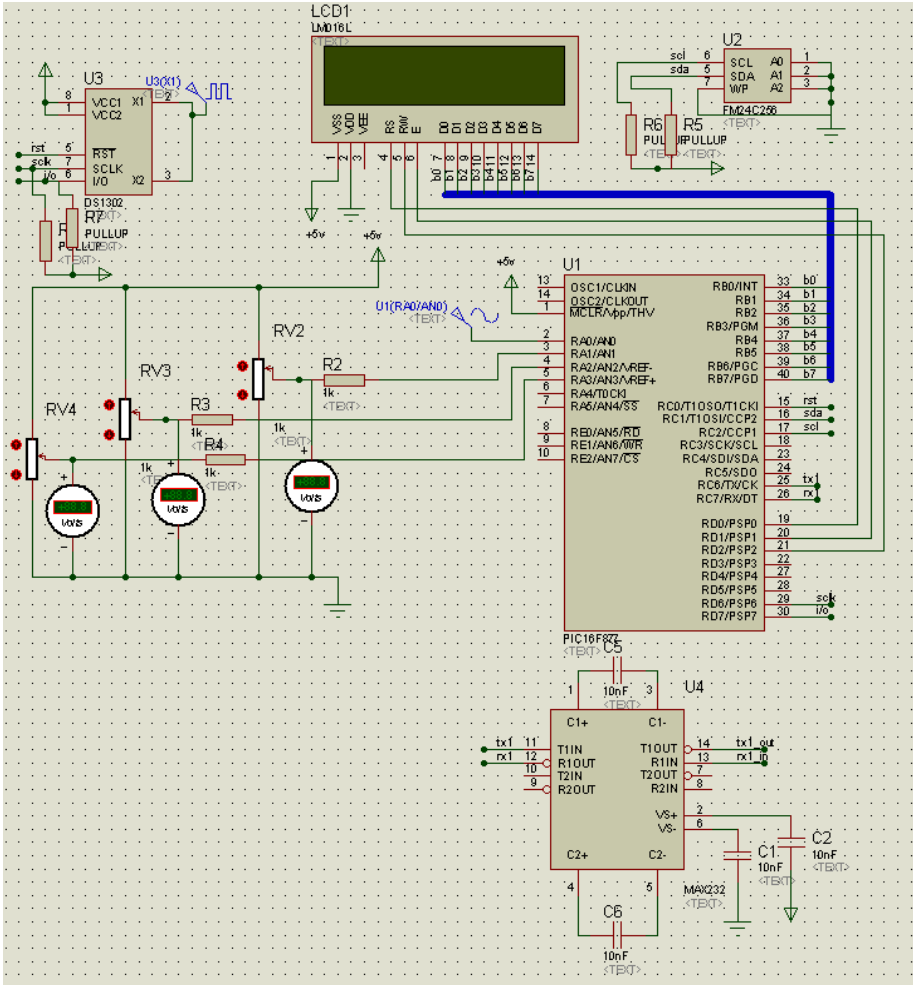

Figura 57. Esquema da ligação física do hardware da aplicação Monitoramento de Dados Remotos

O software da aplicação Cronômetro também foi desenvolvido com a utilização do compilador C PCWH da CCS. O software da aplicação foi testado e validado somente na ferramenta de simulação PROTEUS/ISIS, apresentado na Figura 57. Grande parte do software desta aplicação foi reaproveitado do trabalho desenvolvido por Carlos Cimolin (2004), principalmente os

drivers dos componentes, visando a agilização do processo do mesmo. Alguns códigos são apresentados a seguir.

A função de leitura dos sensores é visualizada a seguir na Figura 26.

```
int16 read_sensors(byte canal)
{
 int16 val16,valor;
   set_adc_channel(canal);                      // escolhe o canal de leitura
   valor = read_adc();                         // efetua a conversão A/D
   delay_ms(20);
  if (valor) valor += 1;
   val16 = (valor * 4) + ((int32)valor * 113)/128;
  return (val16);
}
```

Figura 58. Código da função que efetua a leitura dos sensores

Fonte: Cimolin (2004)

A comunicação RS232 utilizada foi implementada conforme é mostrado abaixo. A função de transmissão pode ser visualizada na Figura 59 e a função de recepção mostrada na Figura 60.

```
void usart_transmite (char dado)
{
 while (!txsta.trmt);        // aguarda o buffer de transmissão esvaziar
 txreg = dado;               // coloca novo caractere para transmissão
}
```

Figura 59. Trecho de código da função transmite dados

Fonte: Cimolin (2004)

```
char usart_recebe (void)
{
 while (!flag_rc);       // aguarda a recepção de caracteres
 return rxreg;           // retorna o caractere recebido
}
```

Figura 60.Trecho de código da função de recepção de dados

Fonte: Cimolin (2004)

O trecho de código apresentado na Figura 61, representa o código de controle da aplicação que efetua a varredura nos sensores.

```
(...)
/****************** Efetua Leitura dos Sensores*****************************/
   for(i=0;i < num_sensores; i++){
          amostra [i] = lê_Dados_Sensor(i);
      }
/****************** Grava dados na memória externa**************************/
    end_escrita = 2*conta_amostra*num_sensores;

   for(i=0;i<num_sensores;i++){
    write_eeprom_ext(end_escrita++, amostra [i]>>8);
    write_eeprom_ext(end_escrita++, amostra [i]);
      }
    conta_amostra++;
(...)
```

Figura 61. Trecho de código da aplicação Monitoramento de Dados Remotos
Fonte: Cimolin (2004)

3.4. DEFINIÇÃO DAS MEDIDAS PARA ANÁLISE

Com o estudo efetuado na seção Medidas de Software, foi observado que a medida de um software dispõe de poucos fatores físicos para o diagnóstico de um resultado preciso. As medidas de software mais utilizadas hoje em dia baseiam-se em dados subjetivos que são distribuídos de acordo com a percepção do indivíduo que o avalia, como, por exemplo, o nível de complexidade que determinada função possui dentro do sistema.

Um software não pode ser avaliado como um todo, ou seja, é necessário dividí-lo em partes dando um peso diferenciado a cada parte. Assim como ocorre com o software, foi observado neste estudo, que, devido aos diferentes aspectos que cada metodologia abrange, ela também não pode ser avaliada como um todo, e, portanto, afirmar que a metodologia ROPES é melhor que a metodologia WOLF ou vice-versa seria incorreto. O que pode ser extraído dos dados levantados neste trabalho é quão bem determinados procedimentos da metodologia ROPES auxiliarão mais e melhor que o da metodologia WOLF, se aplicados a um sistema de pequeno porte, por exemplo.

As medidas definidas para a avaliação das metodologias neste trabalho são subjetivas, sendo que a base para a aplicação de cada medida é o nível de complexidade do sistema e a percepção adquirida em cada fase através da utilização das metodologias. As medidas definidas estão divididas em três categorias, sendo elas:

- Nível de viabilidade: indica a viabilidade da aplicação de determinada fase da metodologia;

- Nível de complexidade: indica a complexidade na aplicação de determinada fase da metodologia; e
- Nível de agilidade: indica se determinada fase da metodologia agiliza o desenvolvimento da aplicação.

Cada medida pode ser classificada em “baixo” / “médio” / “alto”, para cada fase de projeto, de acordo com o nível de complexidade da aplicação. Sua aplicação é feita da seguinte forma: a fase requisitos da metodologia “ALFA”, por exemplo, possui um nível de viabilidade médio para aplicações de pequeno porte, e alto, para aplicações de médio porte.

3.5. ANÁLISE DAS METODOLOGIAS

O modelo CMMI de nível 2, trabalha com o conceito de repetitividade, utilizando experiências anteriores para efetuar um melhor gerenciamento do processo de desenvolvimento de um software. Com base nesse conceito e com a experiência adquirida durante o desenvolvimento das aplicações, foi elaborada a análise das metodologias através da aplicação das medidas definidas.

Como explicado anteriormente, não se pode avaliar a metodologia como um todo, desta forma, a análise das mesmas foi dividida conforme as fases definidas na fase de Projeto deste trabalho, sendo elas Requisitos e Projeto. As fases de Implementação e Testes, por serem iguais para ambas as metodologias não foram avaliadas.

Cada metodologia foi aplicada seguindo os padrões adotados por seus autores, em alguns casos foi observado que a aplicação de determinados diagramas não agregaram um valor considerável no processo de elaboração do software das aplicações.

Foram implementadas três aplicações, sendo as duas primeiras de pequeno porte e a última de médio porte. Para a análise foram consideradas duas classes de aplicação : pequeno e médio porte.

As análises das metodologias foram tabeladas para facilitar sua leitura e visualização. Cada fase da metodologia foi avaliada de acordo com as medidas definidas, relacionando-as com o porte das aplicações.

Na Tabela 4, é apresentado o quadro do nível de viabilidade das fases de Requisitos e Projeto, de acordo com a metodologia e dividido pela complexidade da aplicação analisada.

Tabela 4. Nível de viabilidade

Fase de Projeto	**Metodologia ROPES**		**Metodologia WOLF**	
	Pequeno Porte	Médio Porte	Pequeno Porte	Médio Porte
Requisitos	baixo	alto	alto	alto
Projeto	médio	alto	médio	alto

Na metodologia ROPES a viabilidade de aplicação da fase de Requisitos para aplicações de pequeno porte, é considerada baixa, pois se a aplicação é muito simples, alguns diagramas não auxiliam na elaboração do software e o processo ficaria mais viável sem tais diagramas. Para uma aplicação de médio porte, cada diagrama agrega valor ao resultado, facilitando a visualização e construção das funções do sistema.

Já na metodologia WOLF, como deixa a cargo do projetista a escolha de usar ou não determinados diagramas, numa aplicação de pequeno porte pode-se optar apenas pela descrição da mesma, agilizando seu processo de desenvolvimento.

A fase de Projeto não apresentou diferenças em termos de viabilidade nas metodologias, visto que independente do nível da aplicação, os diagramas de componentes da metodologia ROPES, e os diagramas de blocos da metodologia WOLF, são necessários. Nas aplicações de pequeno porte foram considerado um nível de viabilidade médio, devido ao fato de determinados artefatos desses diagramas não serem tão necessários, como por exemplo, a definição dos componentes de software, que é efetuada em ambas metodologias.

Na Tabela 5 é apresentado o nível de complexidade na aplicação das fases das metodologias. Essa avaliação é subjetiva e foi baseada na experiência adquirida durante o desenvolvimento do trabalho.

Tabela 5. Nível de complexidade

Fase de Projeto	**Metodologia ROPES**		**Metodologia WOLF**	
	Pequeno Porte	Médio Porte	Pequeno Porte	Médio Porte
Requisitos	médio	médio	baixo	médio
Projeto	baixo	baixo	baixo	baixo

Na Tabela 6 é apresentado o nível de agilidade que a fase gerou no desenvolvimento do software. Essa avaliação também é subjetiva e foi baseada na experiência adquirida durante o desenvolvimento do trabalho.

Tabela 6. Nível de agilidade

Fase de Projeto	**Metodologia ROPES**		**Metodologia WOLF**	
	Pequeno Porte	Médio Porte	Pequeno Porte	Médio Porte
Requisitos	baixo	baixo	alto	médio
Projeto	médio	médio	alto	médio

Considerando a análise apresentada, conclui-se que para as aplicações desenvolvidas no LSED a metodologia que mais facilita o desenvolvimento, proporcionando os recursos necessários tanto para aplicações de pequeno porte quanto aplicações de médio porte é a Metodologia WOLF.

Como comentado anteriormente não pode-se afirmar que a Metodologia WOLF é melhor que a Metodologia ROPES, ambas tem suas peculiaridades, o grande diferencial encontrado na metodologia WOLF, é sua versatilidade tanto para aplicações de pequeno porte quanto para aplicações de médio porte, devido ao fato da metodologia oferecer a opção de escolher quais diagramas serão utilizados para o desenvolvimento do software.

4. CONCLUSÕES

Neste trabalho foi efetuado o estudo de duas metodologias de desenvolvimento de sistemas embarcados e de técnicas que são utilizadas pelas mesmas. Três aplicações de portes diferentes foram desenvolvidas utilizando os conceitos estudados, tendo como foco principal o processo de desenvolvimento do software, o que gerou três modelos que apresentam a forma de uso das técnicas e metodologias.

As duas primeiras aplicações foram validades no software de simulação PROTEUS e no Kit de desenvolvimento disponível no LSED. Como o foco do estudo e avaliação foi no desenvolvimento do software e devido ao tempo disponível para a realização desse trabalho, optou-se por validar a última aplicação apenas ao nível de simulação.

Devido à falta de familiarização com a área de sistemas embarcados, foram encontradas diversas dúvidas a respeito da correta elaboração dos diagramas utilizados pelas metodologias, em especial o diagrama de estados. Opiniões de profissionais da área foram consultadas para a obtenção do melhor resultado possível. Os diagramas passaram por diversas transformações durante a fase de projeto, gerando várias versões até a obtenção do resultado apresentado neste texto.

O software de cada aplicação foi desenvolvido uma única vez, pois foi observado no decorrer da execução do presente trabalho, que os dados disponíveis para a elaboração do mesmo eram muito parecidos em ambas metodologias.

Durante o desenvolvimento do trabalho proposto foram consolidados conhecimentos na área de sistemas computacionais embarcados, metodologias de projeto para tais sistemas e medidas de software. Foram selecionadas três aplicações de referência para as quais foram executadas etapas de desenvolvimento de software utilizando as técnicas e metodologias estudadas, incluindo o levantamento de requisitos, a especificação, projeto conceitual, implementação e validação

Um levantamento bibliográfico sobre medidas de software foi elaborado na seção Medidas de Software, apresentando as principais medidas utilizadas no desenvolvimento de software para o levantamento de informações. Também foi efetuada uma abordagem a respeito do CMMI adotado por diversas empresas da área, com o objetivo de tornar o processo de desenvolvimento de software mais eficiente e controlado. Utilizando-se dessas referências, juntamente com opiniões de

profissionais da área de engenharia de software foram definidas as medidas utilizadas para a análise das metodologias.

Os modelos implementados foram avaliados através do conjunto de medidas definido. Com base nessa análise foram efetuadas recomendações a respeito das metodologias estudadas, assim como a identificação das fases mais adequadas para as classes de aplicações tipicamente desenvolvidas no âmbito dos projetos de pesquisa executados no LSED.

Por fim, levando-se em conta os resultados obtidos, considera-se que os objetivos estabelecidos na proposta deste trabalho foram alcançados. A seguir algumas sugestões de continuidade deste trabalho são apresentadas.

Foram utilizadas duas metodologias bases para desenvolvimento deste trabalho, que são as metodologias utilizadas pelos pesquisadores do LSED, no entanto, com o passar do tempo novas metodologias são apresentadas, visando facilitar o projeto e agilizar a elaboração dos sistemas embarcados. Considerando isso, novas pesquisas poderão ser realizadas a fim de incrementar o número de metodologias avaliadas confirmando as recomendações efetuadas neste trabalho, e/ou trazendo novas recomendações.

As aplicações utilizadas neste trabalho possuem um nível de complexidade relativamente pequeno, por serem aplicações de pequeno e médio porte. Uma forma de incrementar os dados levantados, aumentando a abrangência desse trabalho, seria a análise e elaboração de uma aplicação com um nível de complexidade maior, utilizando as metodologias apresentadas.

Outro fator que poderá ser avaliado, visando agregar valor aos resultados obtidos, é a análise em conjunto com os fatores externos ao software, em suma, envolver o hardware do sistema nas avaliações e recomendações, visto que este trabalho atuou focado no processo de desenvolvimento de software para sistemas computacionais embarcados.

REFERÊNCIAS BIBLIOGRÁFICAS

ABREU, Fernando B. As métricas na gestão de projetos de desenvolvimento de sistemas de informação. In: JORNADAS PARA A QUALIDADE NO SOFTWARE, Lisboa, 1992. **Anais...** [S.l.: s.n.], 1992.

ANIDO, M, L. **Sistemas embutidos**. Disponível em: <http://labase.nce.ufrj.br/sistemas _embutidos/> Acesso em: 28 abr 2005.

BARTIE, Alexandre. **Garantia da Qualidade de Software**: As melhores práticas de engenharia de software aplicadas à sua empresa. Campus, 2001. 290p.

BOMFIM, F; AZEVEDO M; HUDSON S. **Métricas de software on-line**. Disponível em: <http://www.internext.com.br/mssa/ medidas.html> Acesso em: 20 maio 2005.

CARRO, L.; WAGNER, F.R. **Sistemas computacionais embarcados**. In: JAI'03 – XXII Jornadas de Atualização em Informática, Capítulo 2, ago 2003.

CIMOLIN, Carlos. **Desenvolvimento de uma plataforma de sistema computacional embarcado para monitoramento de dados remotos**, 2004. 78p. (Trabalho de Conclusão de Curso). Graduação em Ciência da Computação, Universidade do Vale do Itajaí.

CNZ Engenharia. **Programa embedded software**. Cotia: CNZ Engenharia. Disponível em: <http://www.dimap.ufrn.br/~ivan/STRE.html> Acesso em 28 abr 2005.

CORDEIRO, Marco A. Métricas de Software. **Bate Byte**, n.101, Set. 2000. Disponível em: <http://www.pr.gov.br/batebyte/edicoes/2000/bb101/metricas.htm >. Acesso em: 28 abr 2005.

DOUGLASS, Bruce Powel. **Real-Time UML second edition**: Developing Efficient Objects for embedded systems. Reading: Addison-Wesley, 2000. 327p.

ESMIN, Ahmed Ali Abdalla. **Modelando com UML**. Disponível em: <http://www.dcc.ufla.br/infocomp/artigos/v1.1/tutorialuml.pdf>. Acesso em 28 abr 2005.

JACOBSON, Ivar; BOOCH Grady; RUMBAUGH James. **UML** guia do usuário. O mais avançado tutorial sobre Unified Modeling Language (UML), elaborado pelos próprios criadores da linguagem. Rio de Janeiro: Campus, 2000.

MACEDO, Marcus V. R. **Uma proposta de aplicação da métrica de pontos de função em aplicações de dispositivos portáteis**, 2003. Dissertação (Mestrado)–Curso de Pós-Graduação em Computação, Instituto de Computação. Universidade Estadual de Campinas, 2003.

MARTIN, Grant. **UML for embedded systems specification and design**: Motivation and Overview. Cadence Design Systems. CA, 2002.

MORAES, Fernando; CALAZANS, Ney; MARCON, César; MELLO, Aline. Um ambiente de compilação e simulação para processadores embarcados parametrizáveis. In: WORKSHOP IBERCHIP, 7., 2001, Montevidéu. **Memorias…** [S.l.: s.n.], 2001. 11 p.

OYAMADA, M. S; WAGNER, F. R. **Co-simulação de sistemas eletrônicos embarcados**. In: SEMANA ACADÊMICA DO PPGC, 4.,1999, Porto Alegre. **Anais...** Porto Alegre: PPGC da UFRGS, 1999.

PORTO, Alessandra; LARDIZABAL, Gunther. **UML:** concorrência e tempo-real. 2000. Disponível em: <http://www.inf.ufsc.br/~porto/uml/grupo4.htm>. Acesso em: 16 maio 2005.

PRESSMAN, Roger. **Engenharia de Software**. São Paulo: Makron Books, 1995.

SEI Software Engineering Institute. **CMMI Capability Maturity Model Integration**. Disponível em: <http://www.sei.cmu.edu>. Acesso em 15 out 2005.

SILBERSCHATZ. Galvin. **Sistemas Operacionais**: conceitos. São Paulo: Prentice Hall, 2000.

SHAW, Alan C. **Sistemas e software de tempo-real.** Porto Alegre: Bookman, 2003. 240p.

SOUZA, D.J. **Desbravando o PIC**. 1.ed., São Paulo: Érica, 2000.

ZELENOVSKY, R; MENDONÇA, A. **Introdução aos sistemas embutidos.** Disponível em: <http://www.mzeditora.com.br/artigos/embut.htm>. Acesso em: 28 abr 2005.

WOLF, Wayne. **Computer as components**: principles of embedded computing system design. São Francisco: Morgan Kaufman, 2001.

WOLF, Wayne. What is embedded computing? **IEEE Computer**, v.35, n.1 p.136-137, jan. 2002.